



CS 540 Introduction to Artificial Intelligence

Neural Networks (II)

University of Wisconsin-Madison

Fall 2022

Today's outline

- Single-layer Perceptron Review
- Multi-layer Perceptron
 - Single output
 - Multiple output
- How to train neural networks
 - Gradient descent

Review: Perceptron

- Given input \mathbf{x} , weight \mathbf{w} and bias b , perceptron outputs:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

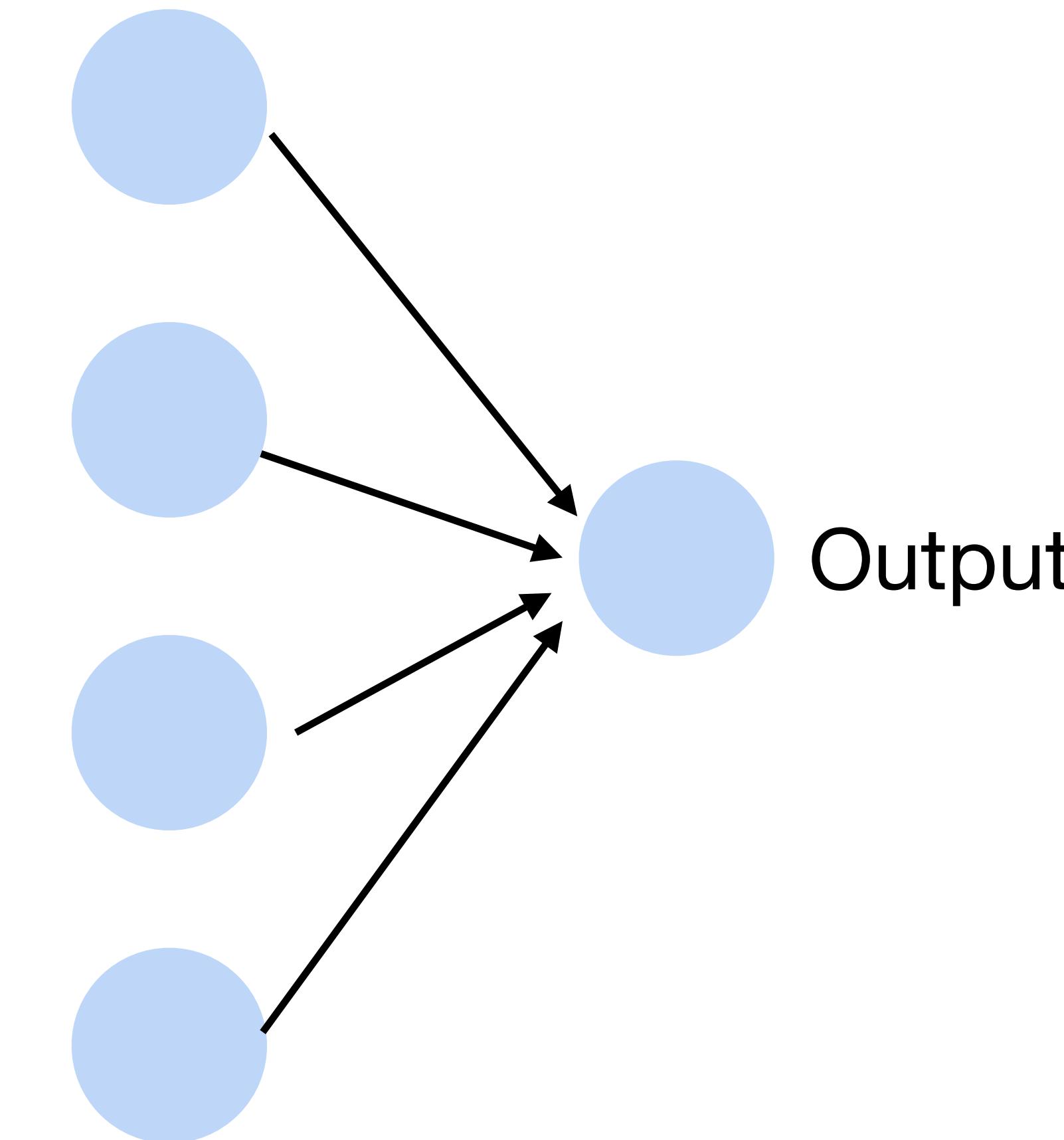
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation function

Cats vs. dogs?

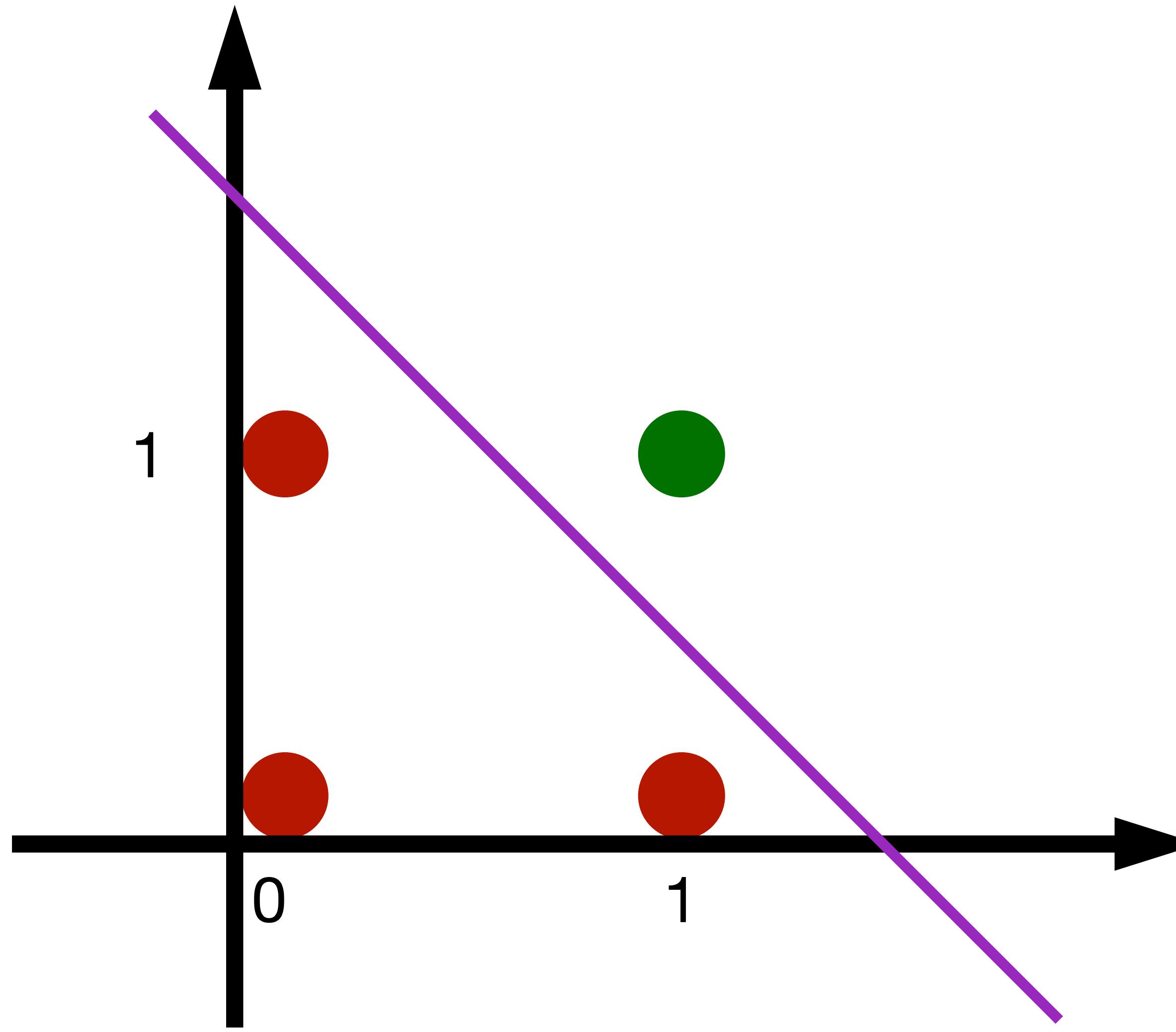


Input



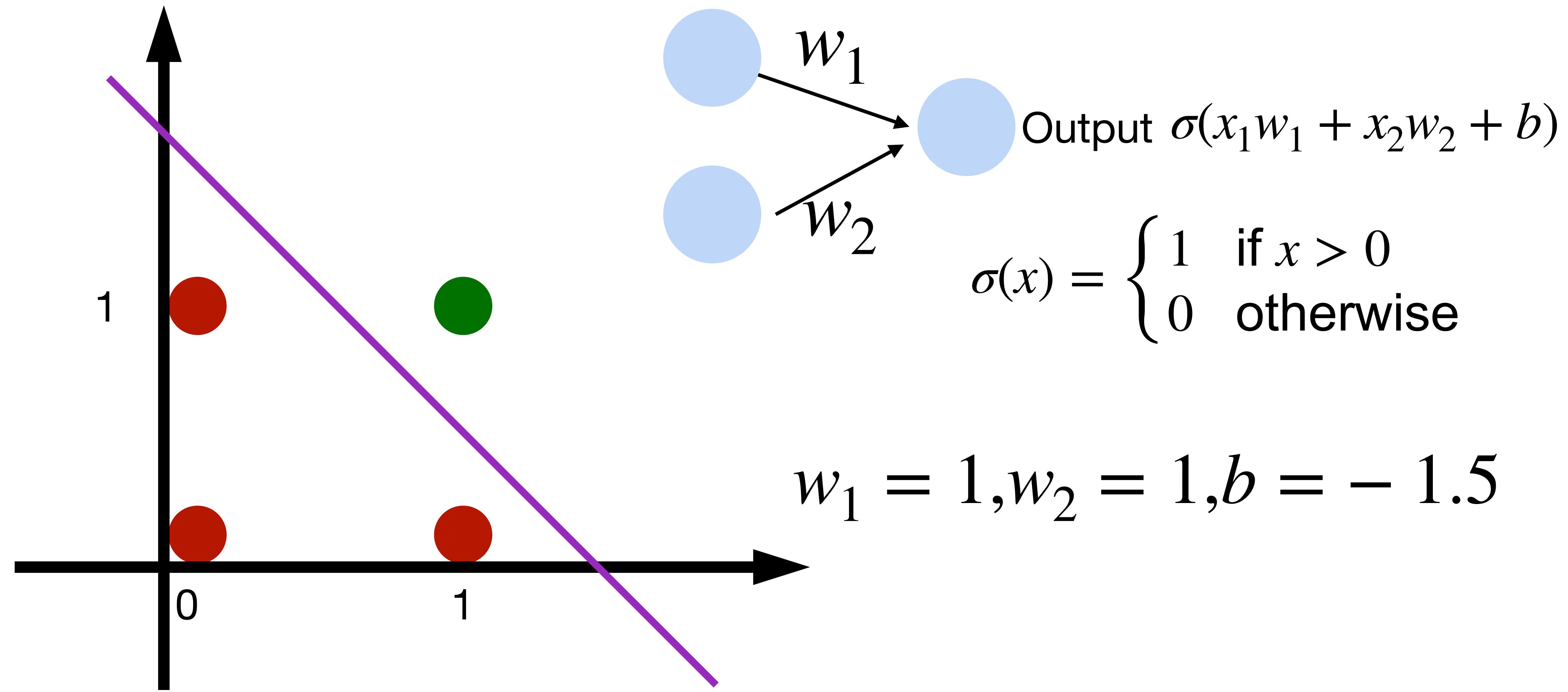
Learning AND function using perceptron

The perceptron can learn an AND function



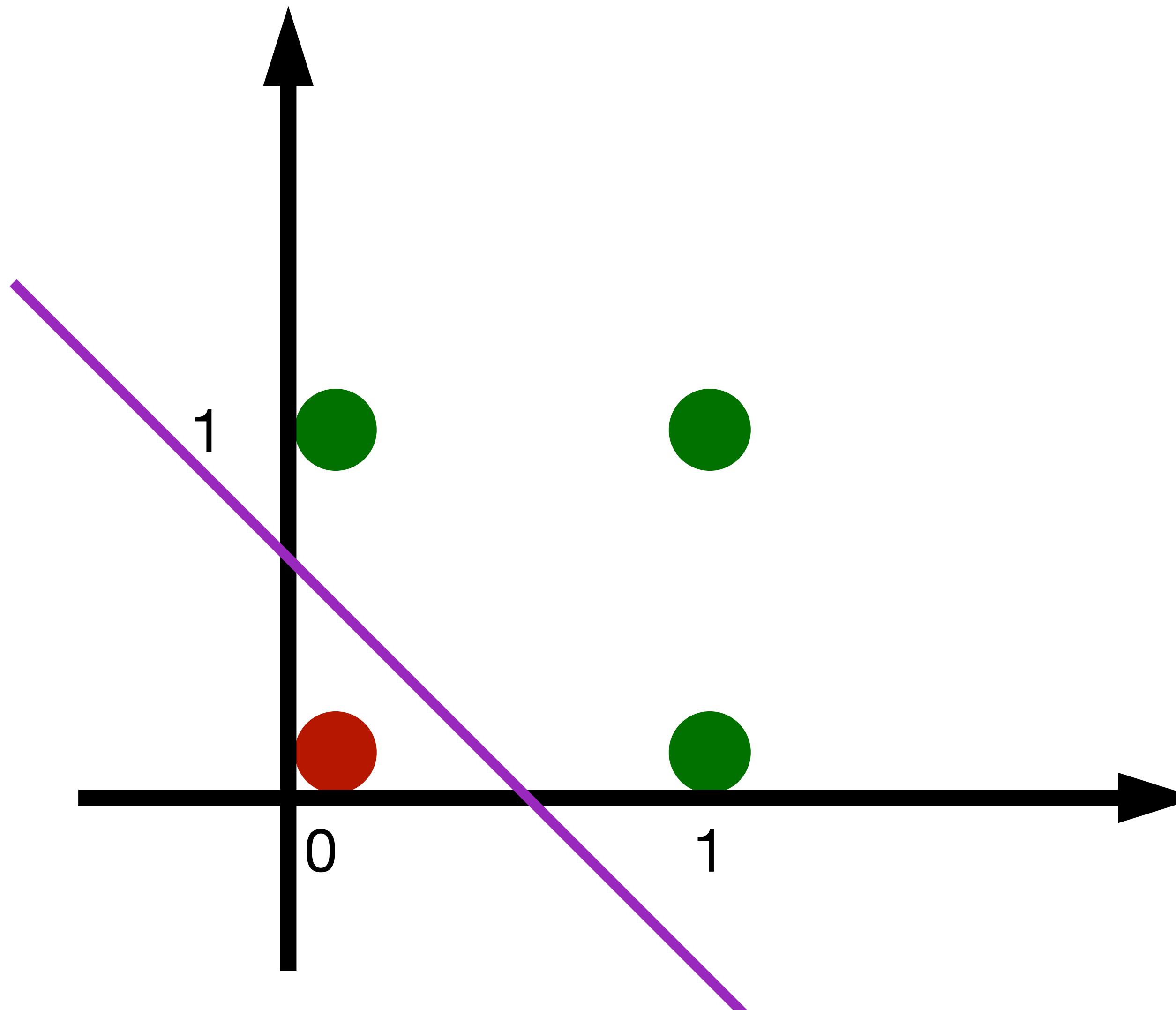
Learning AND function using perceptron

The perceptron can learn an AND function



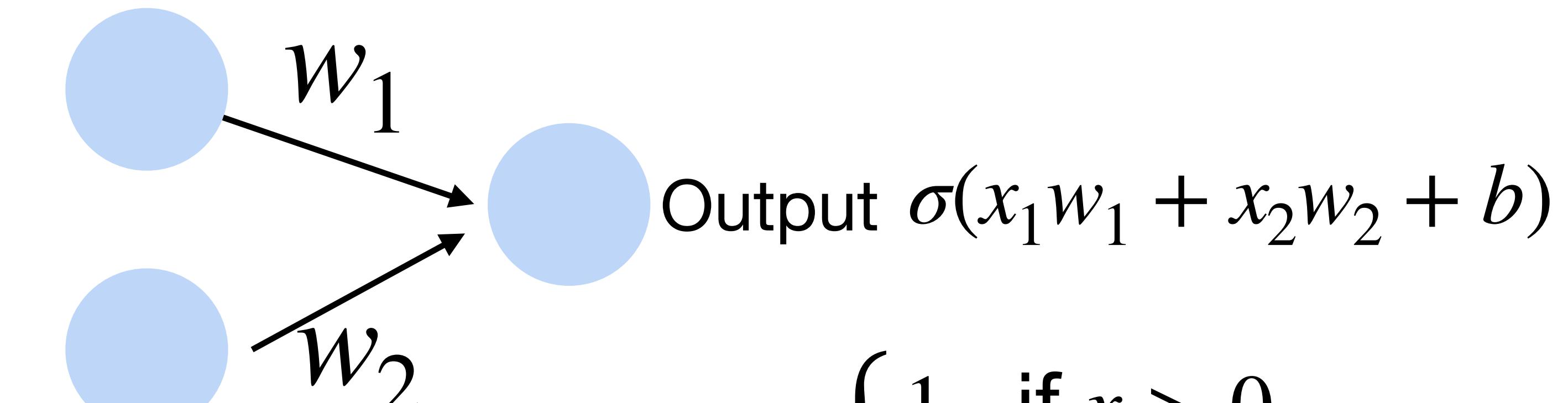
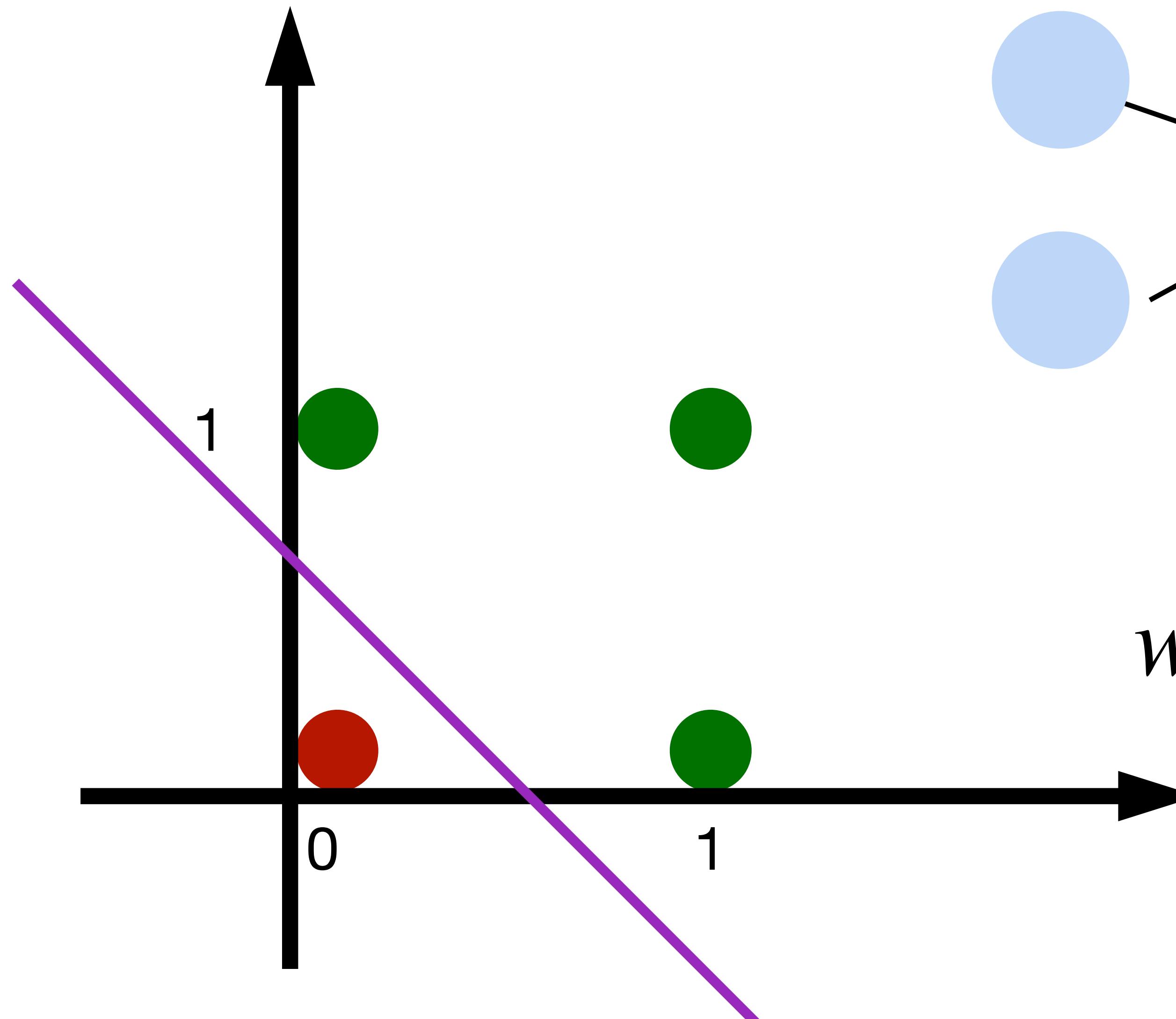
Learning OR function using perceptron

The perceptron can learn an OR function



Learning OR function using perceptron

The perceptron can learn an OR function



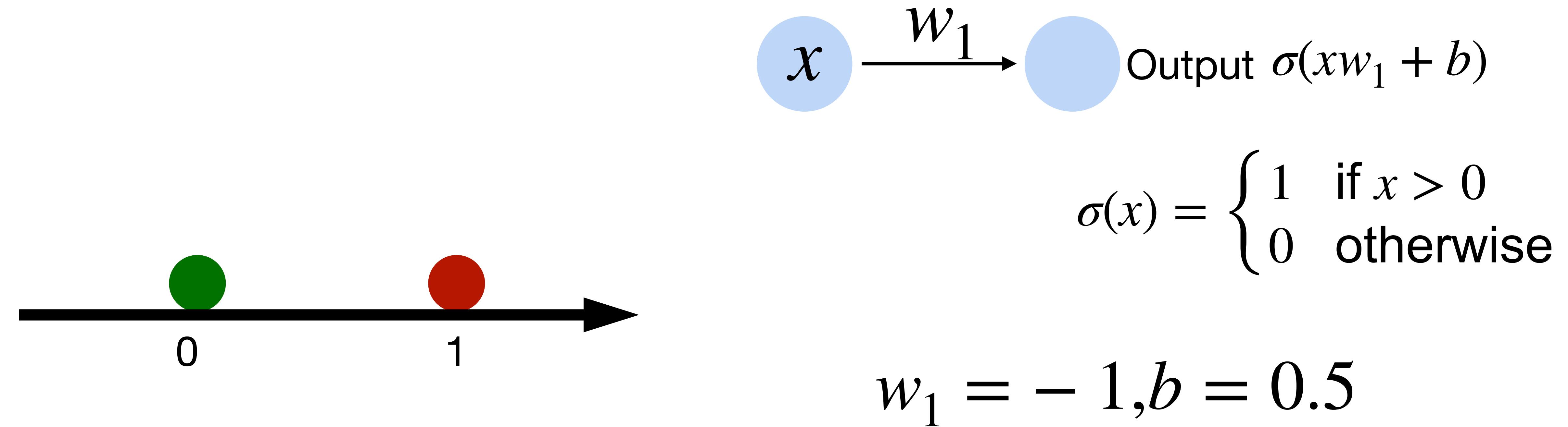
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -0.5$$

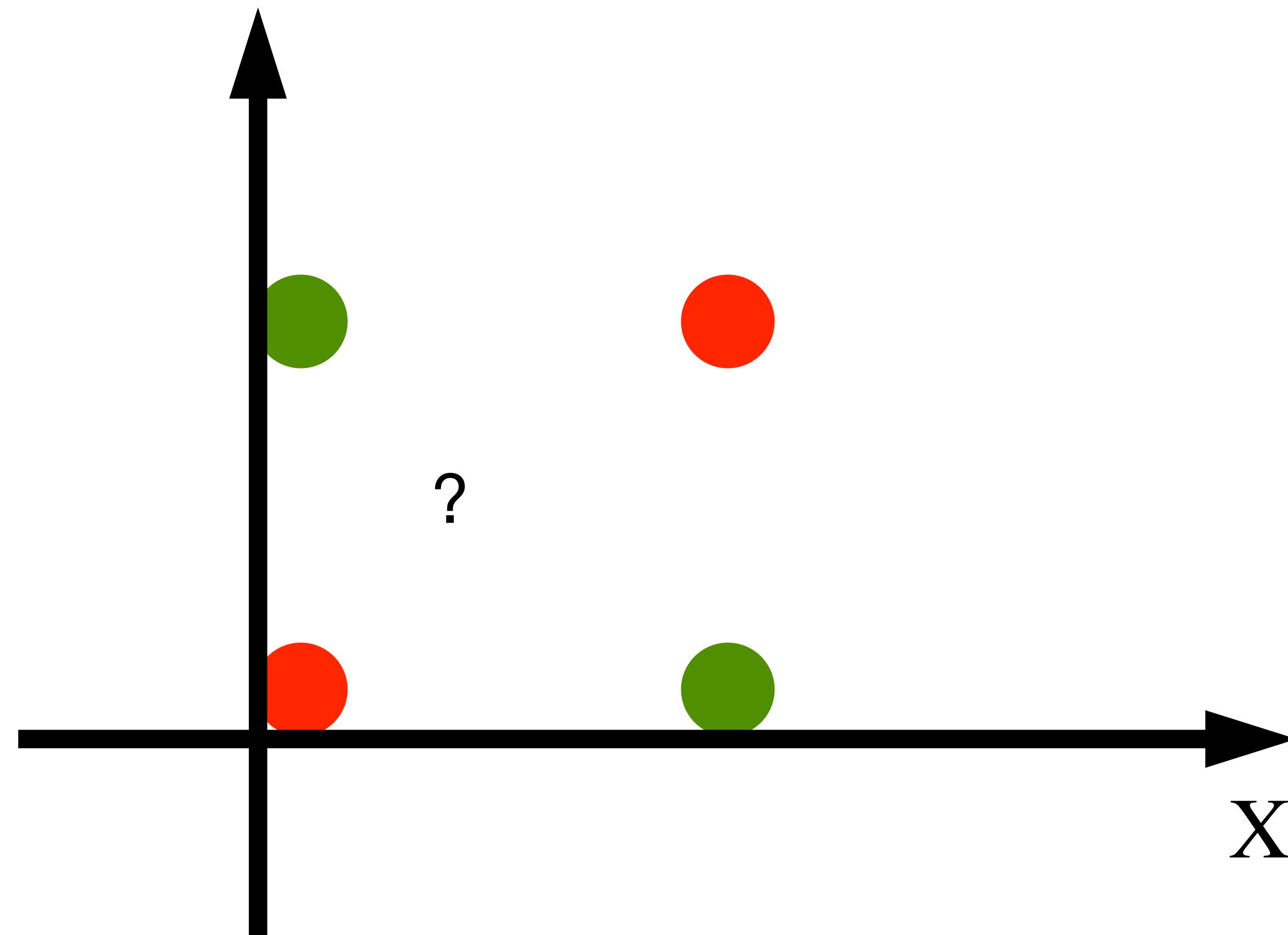
Learning NOT function using perceptron

The perceptron can learn NOT function (single input)



The limited power of a single neuron

The perceptron cannot learn an **XOR** function
(neurons can only generate linear separators)



$$x_1 = 1, x_2 = 1, y = 0$$

$$x_1 = 1, x_2 = 0, y = 1$$

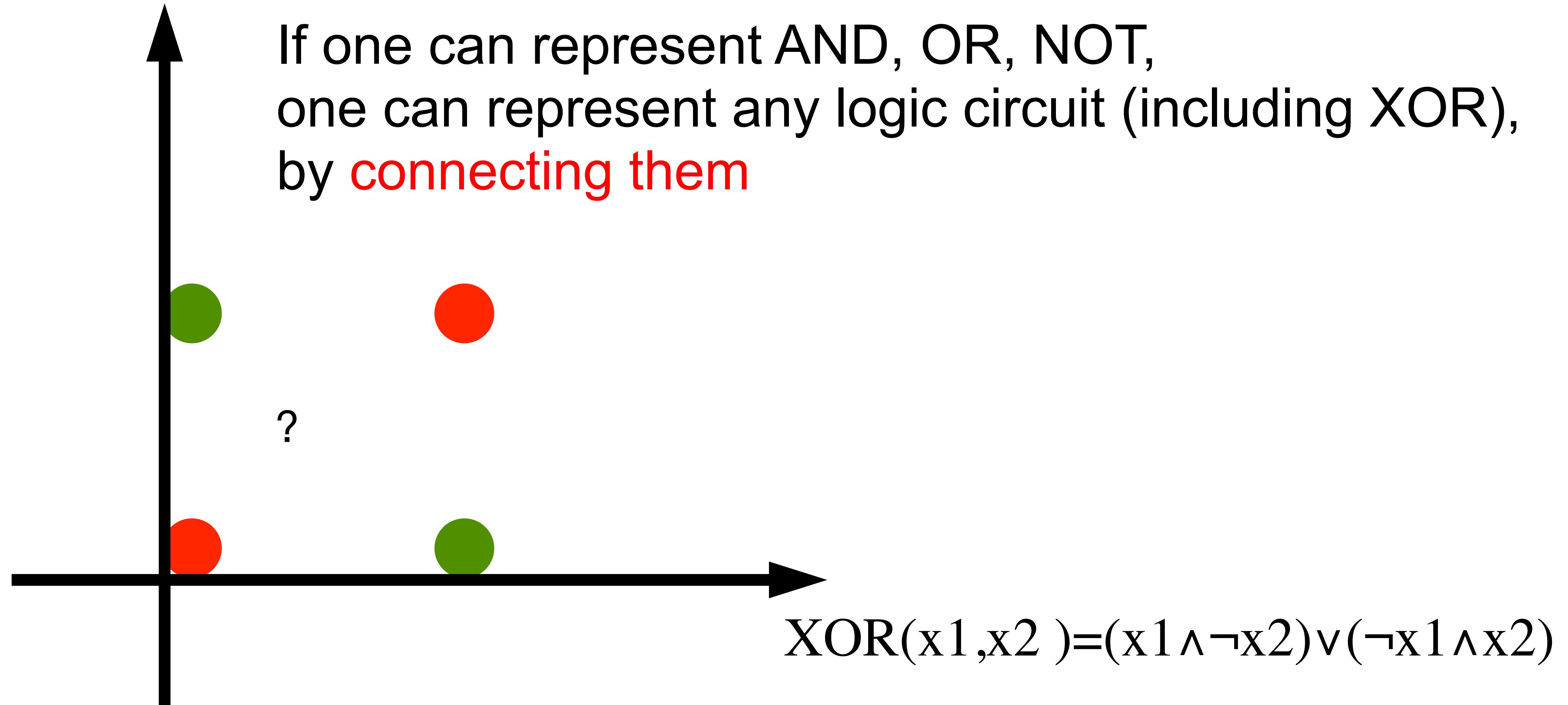
$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$

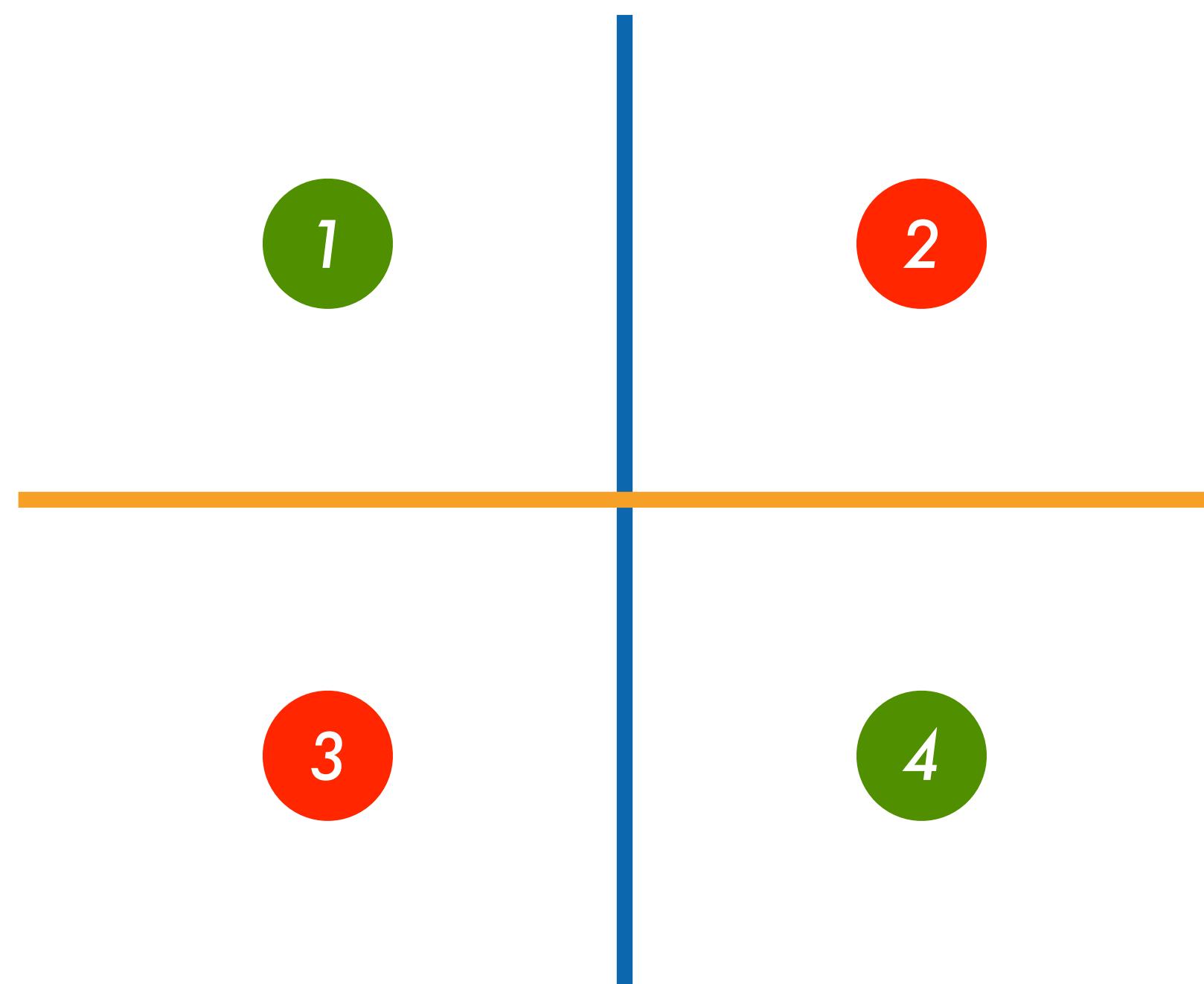
$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

The limited power of a single neuron

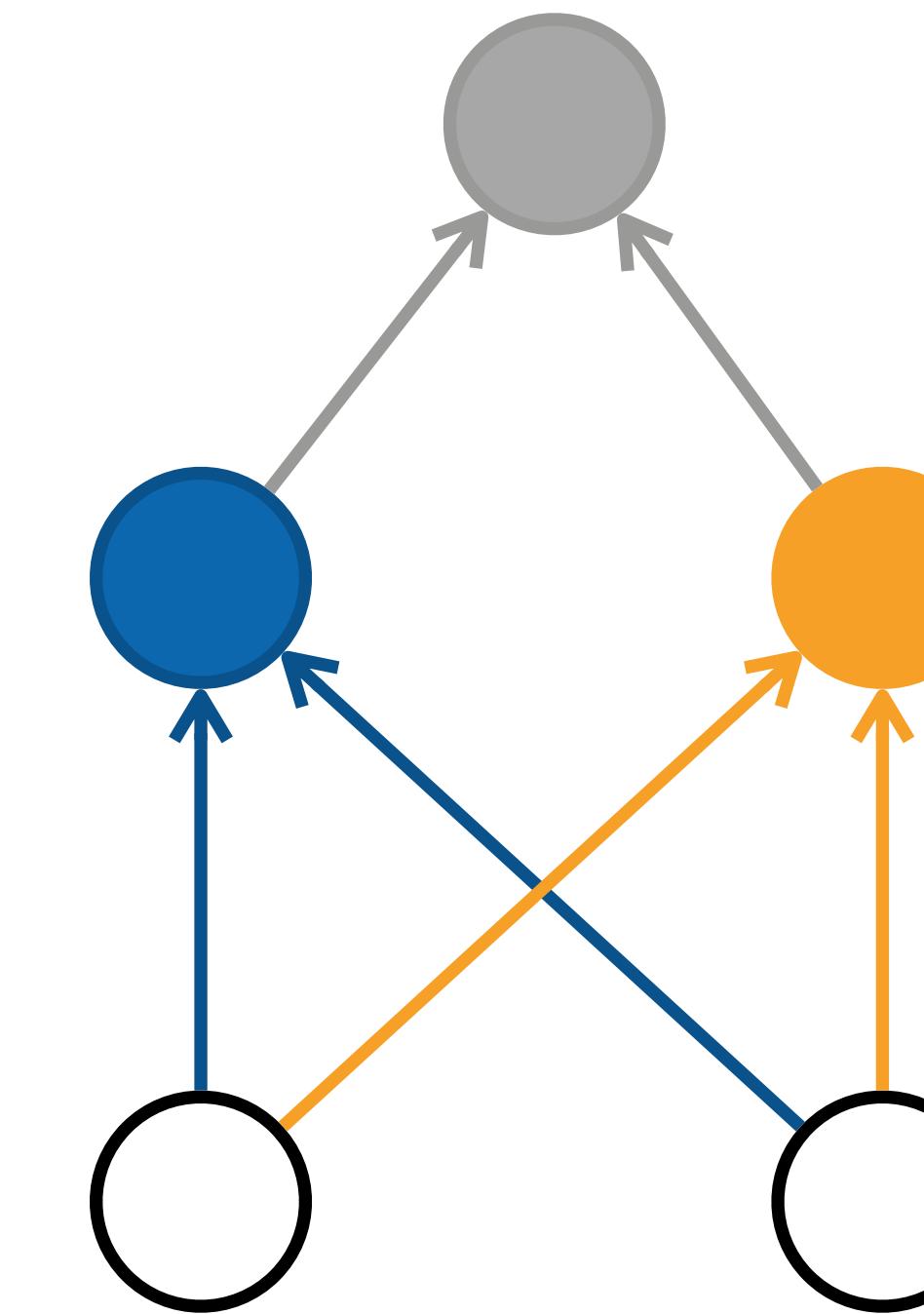
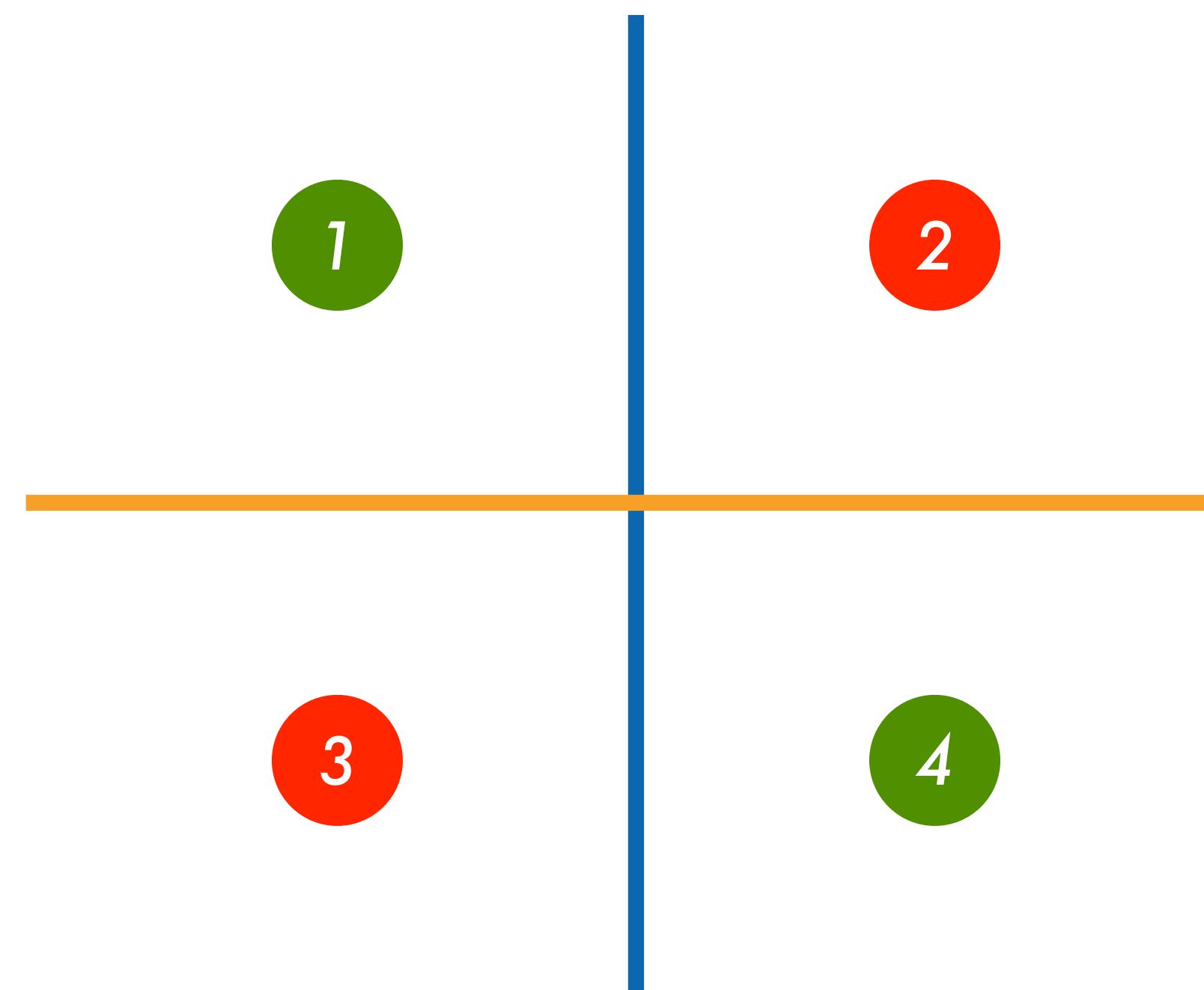
XOR problem



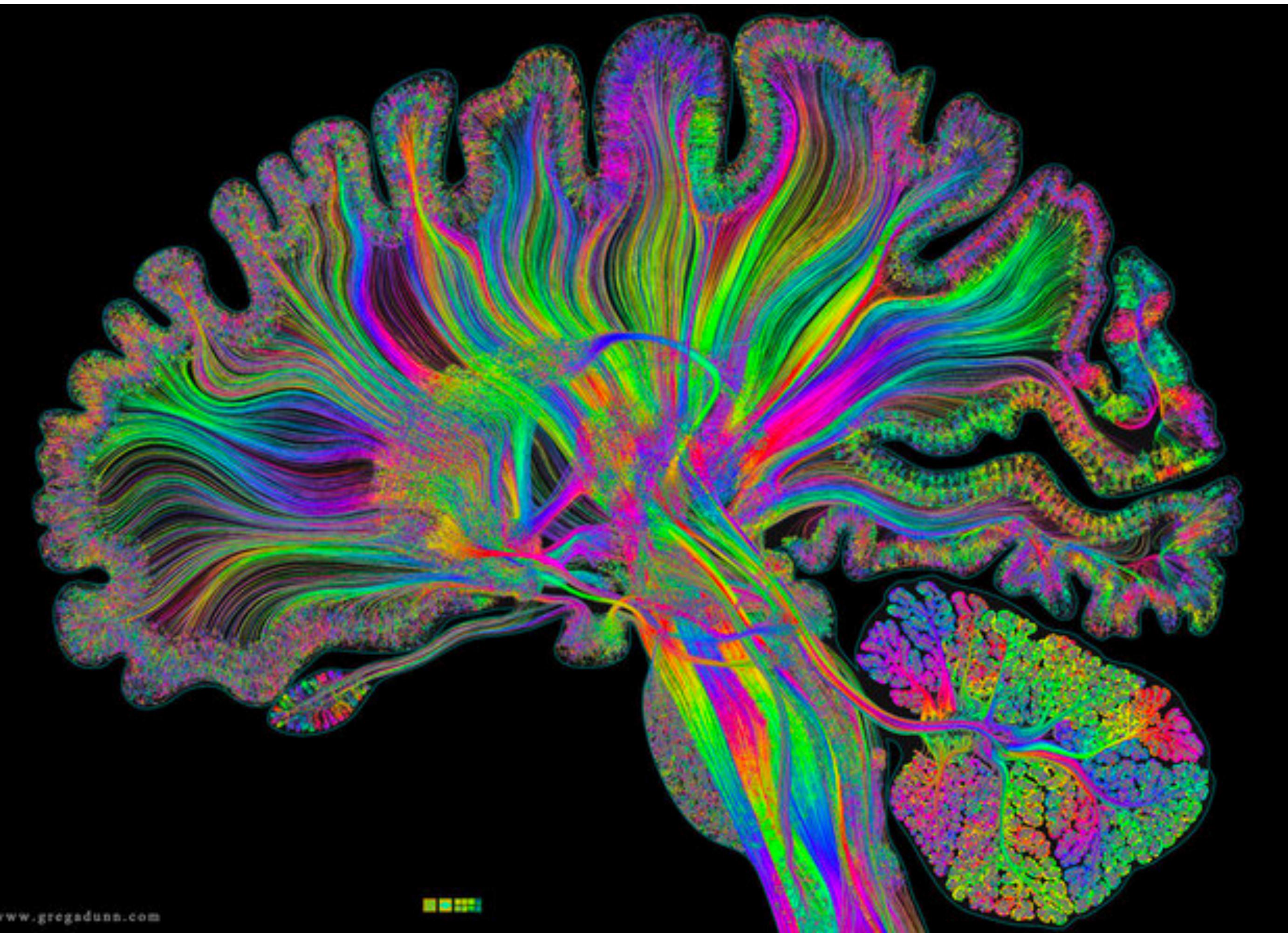
Learning XOR



Learning XOR

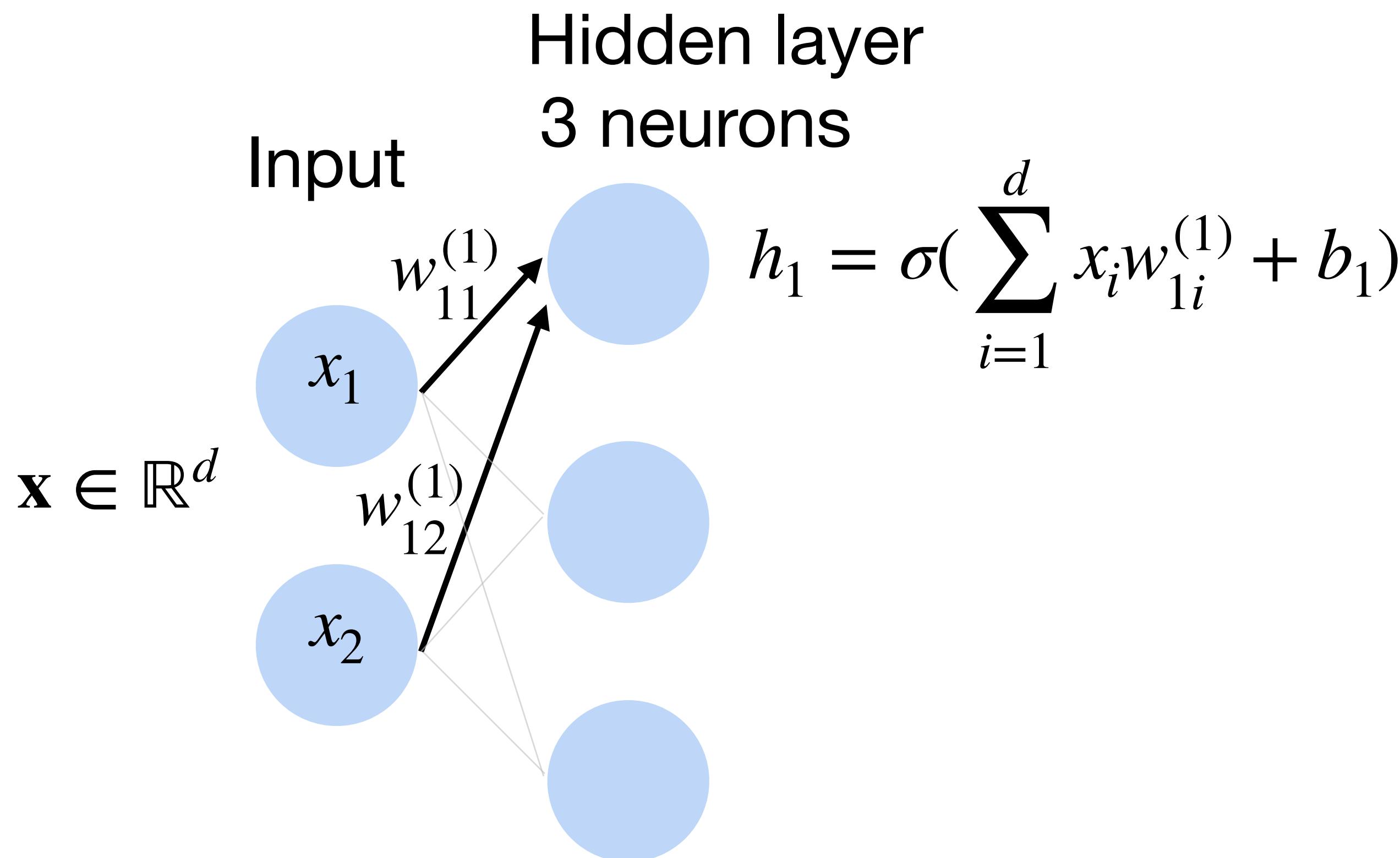


Multilayer Perceptron



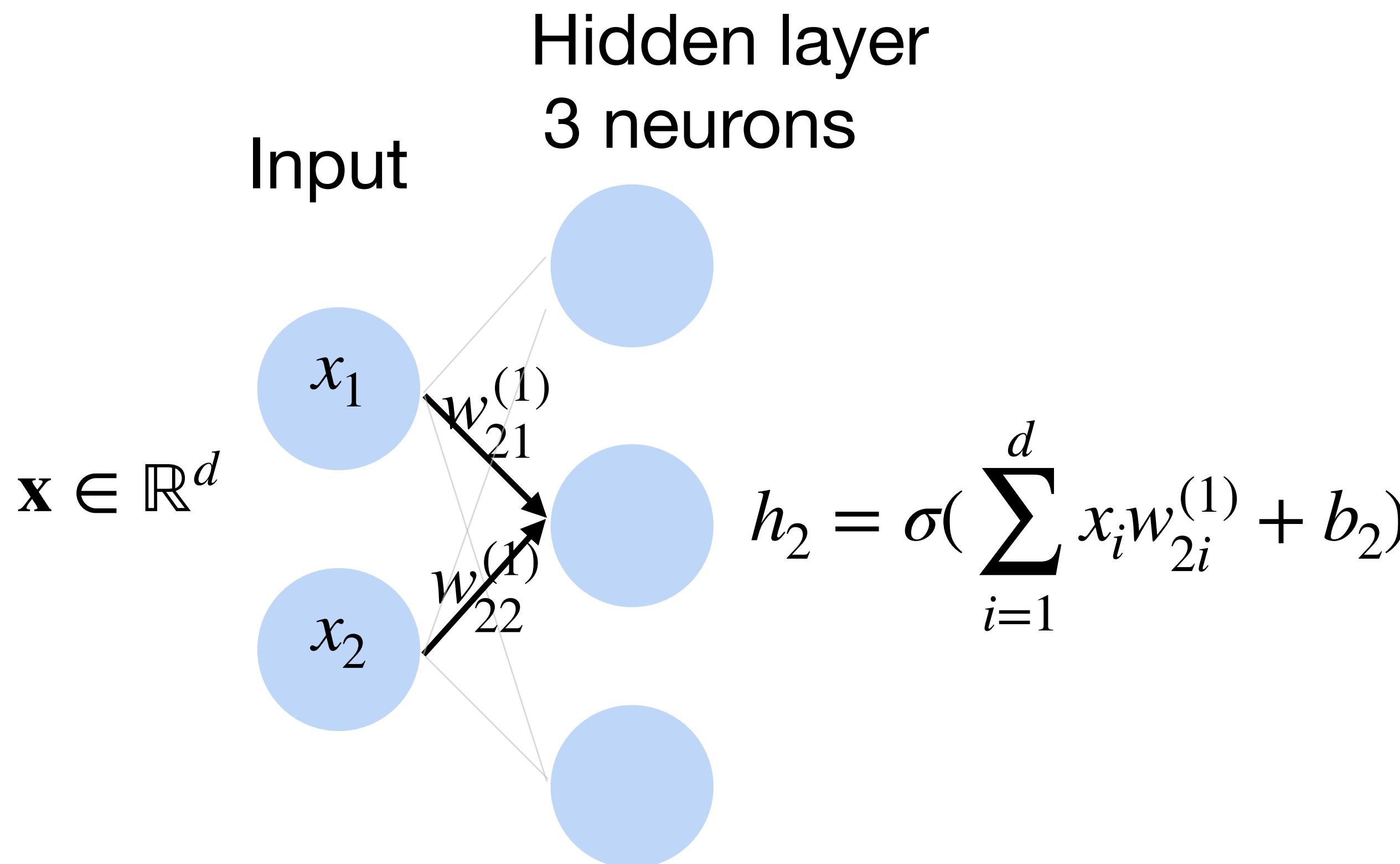
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



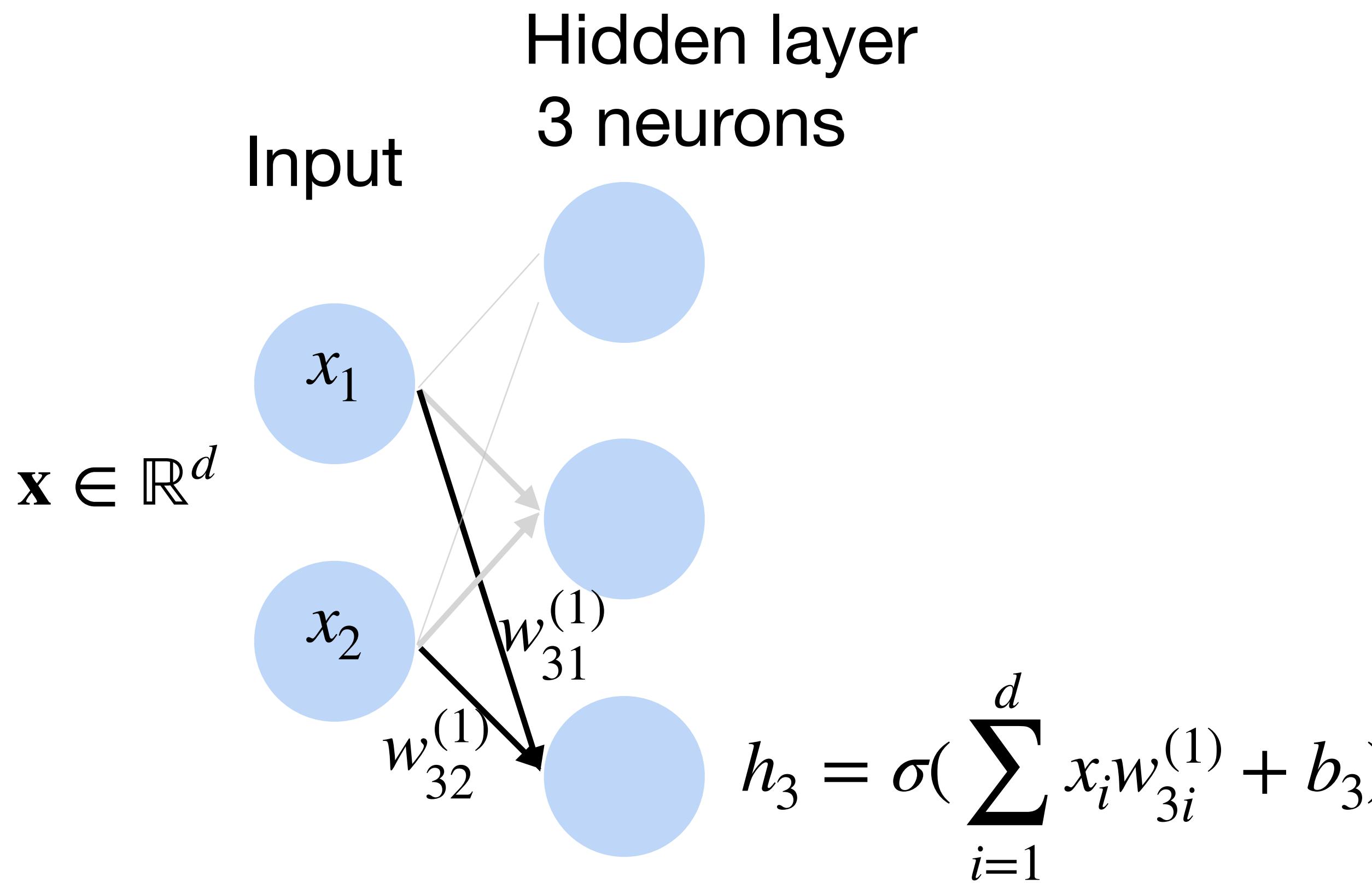
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



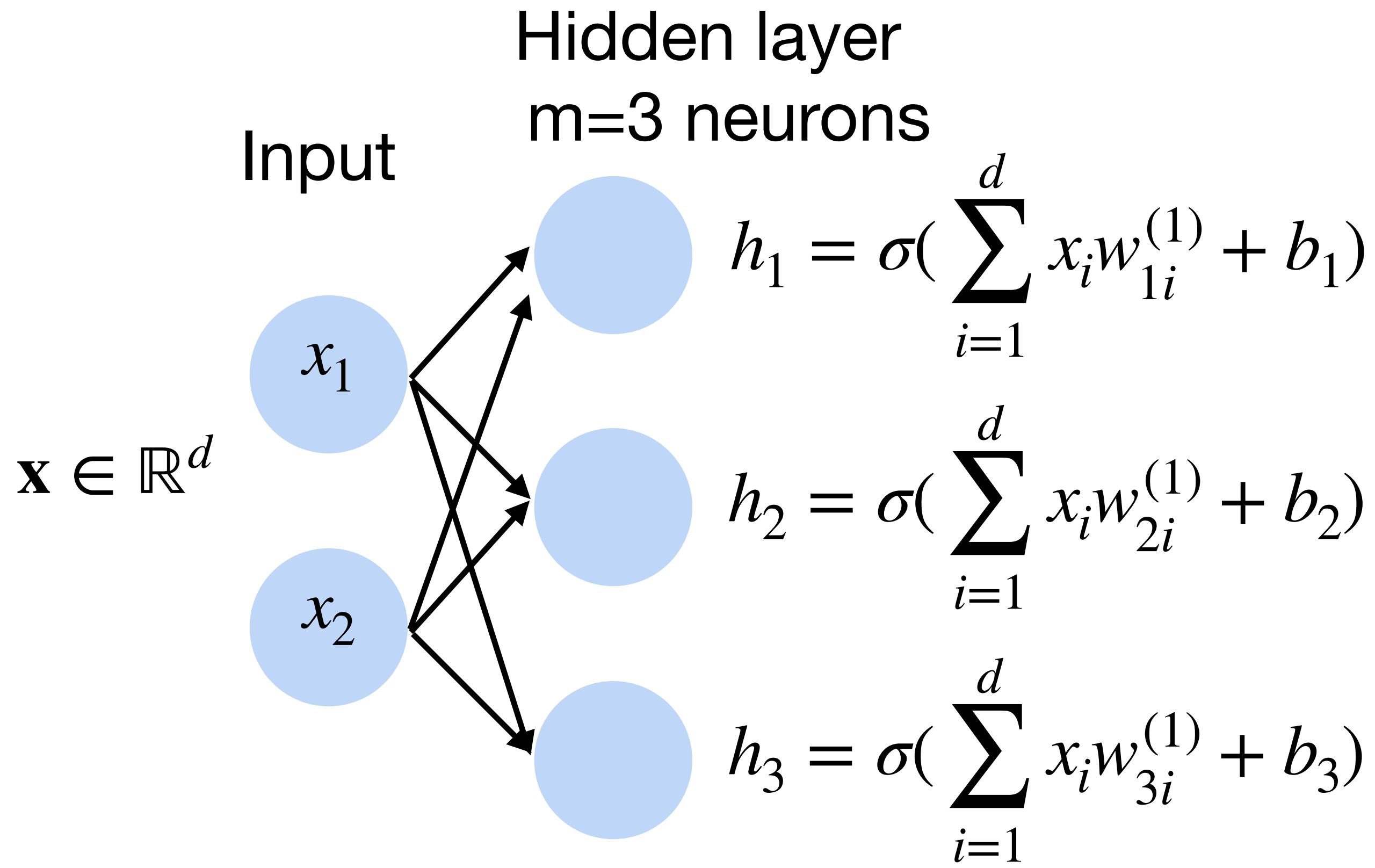
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



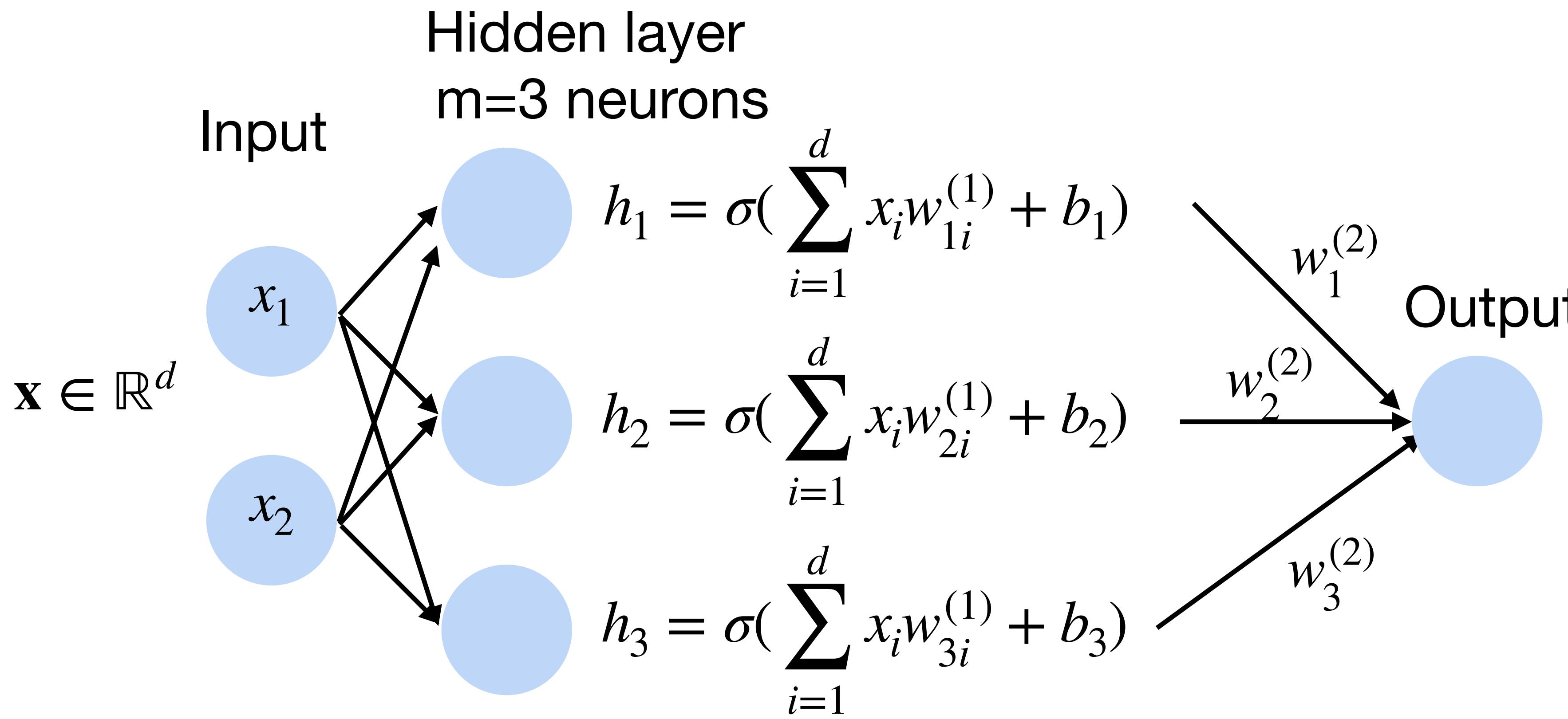
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



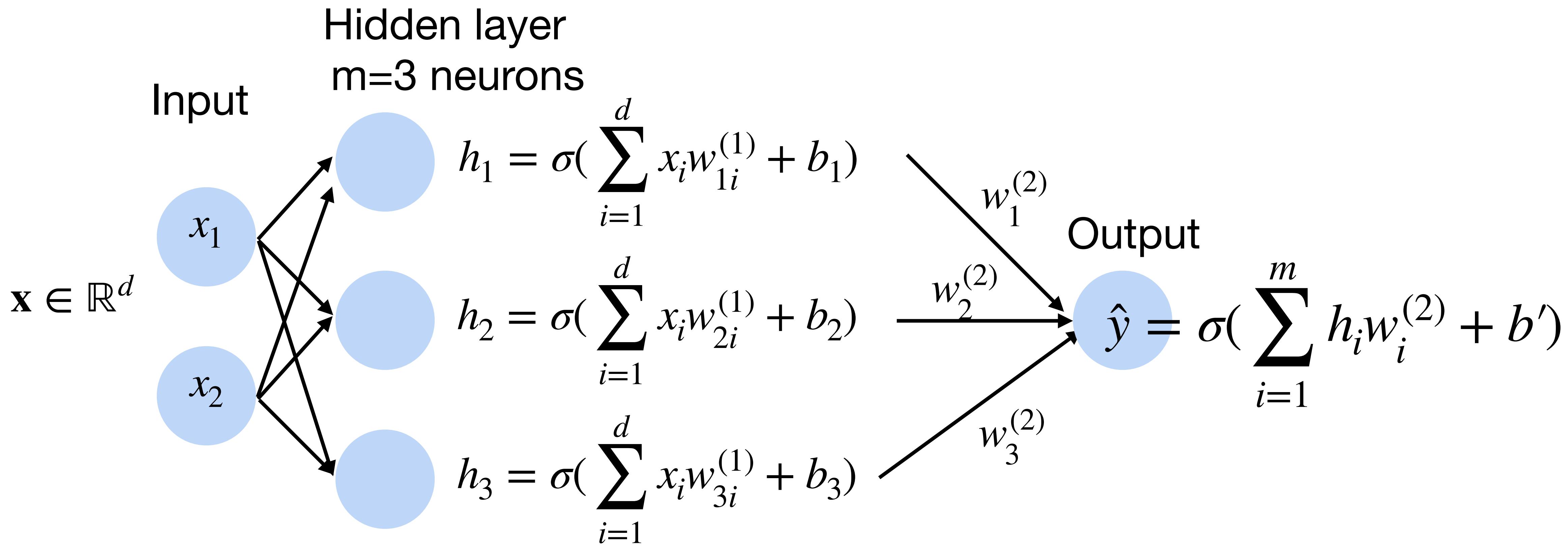
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



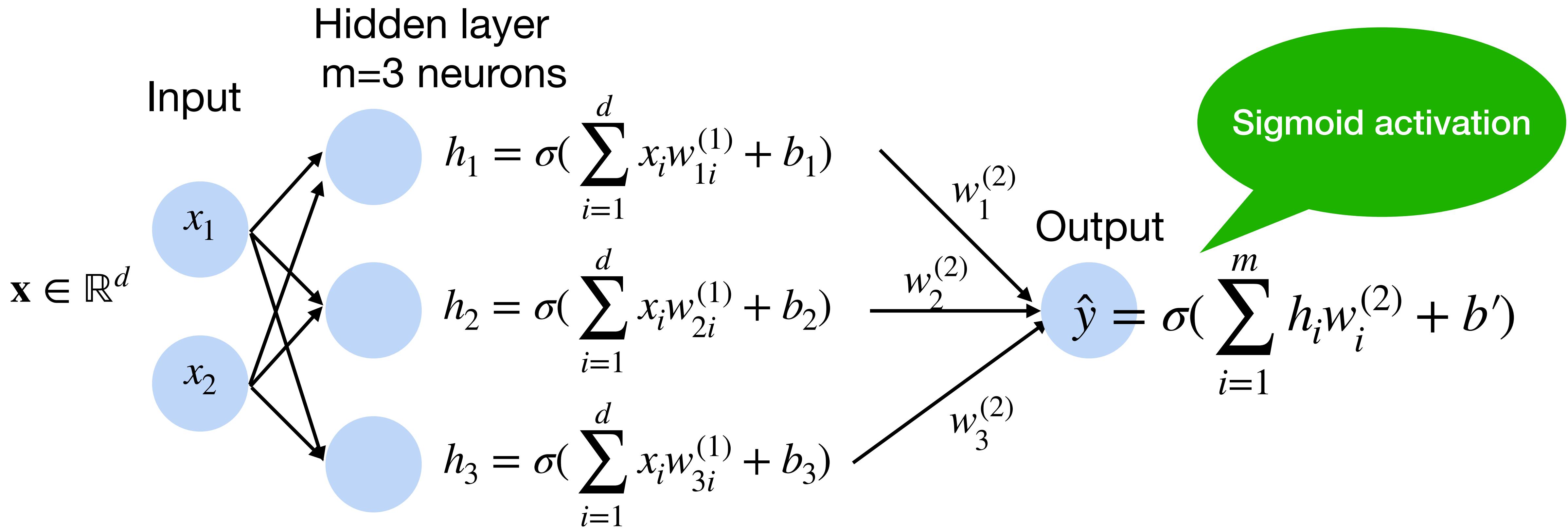
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



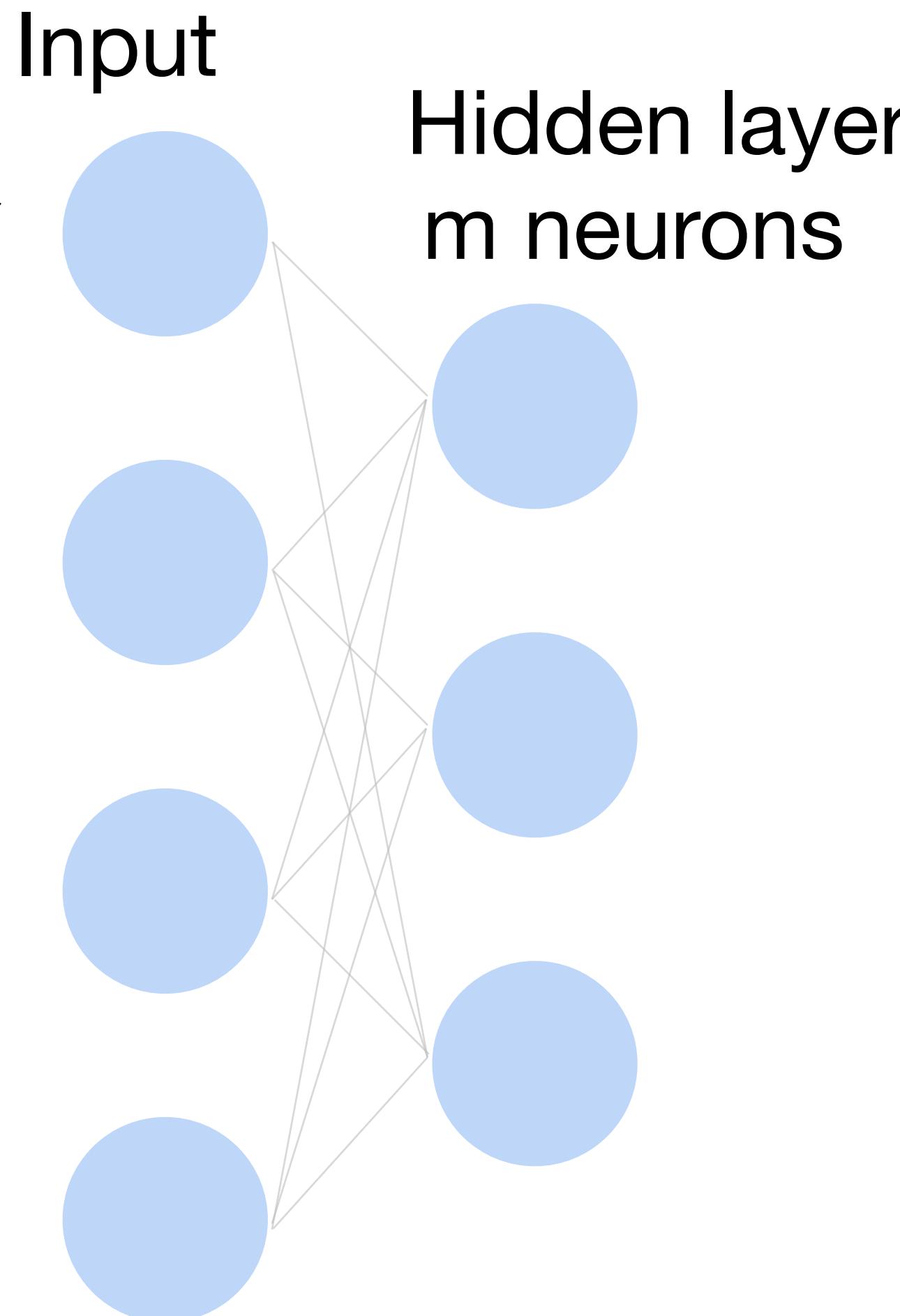
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



Multi-layer perceptron: Matrix Notation

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

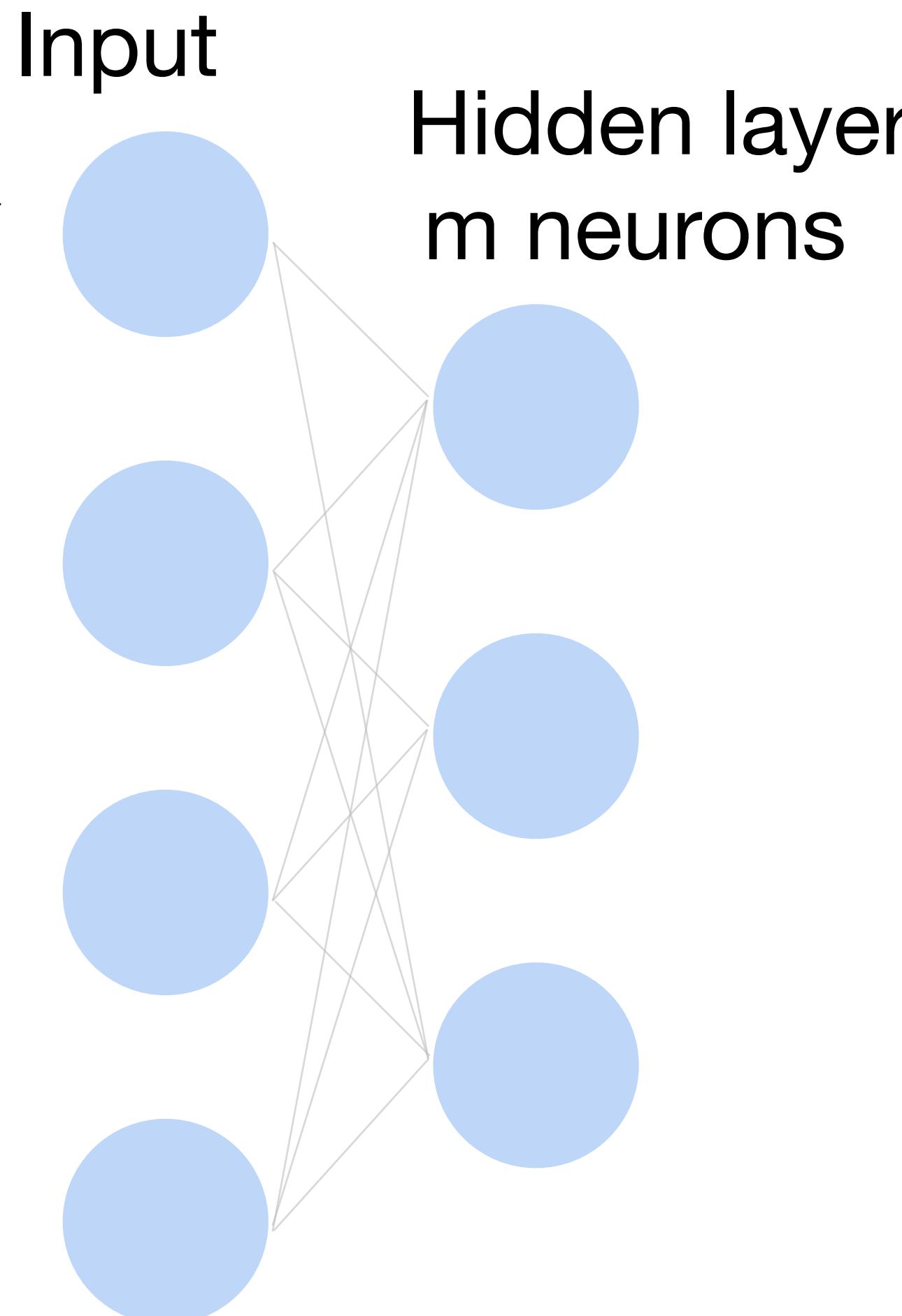


Multi-layer perceptron: Matrix Notation

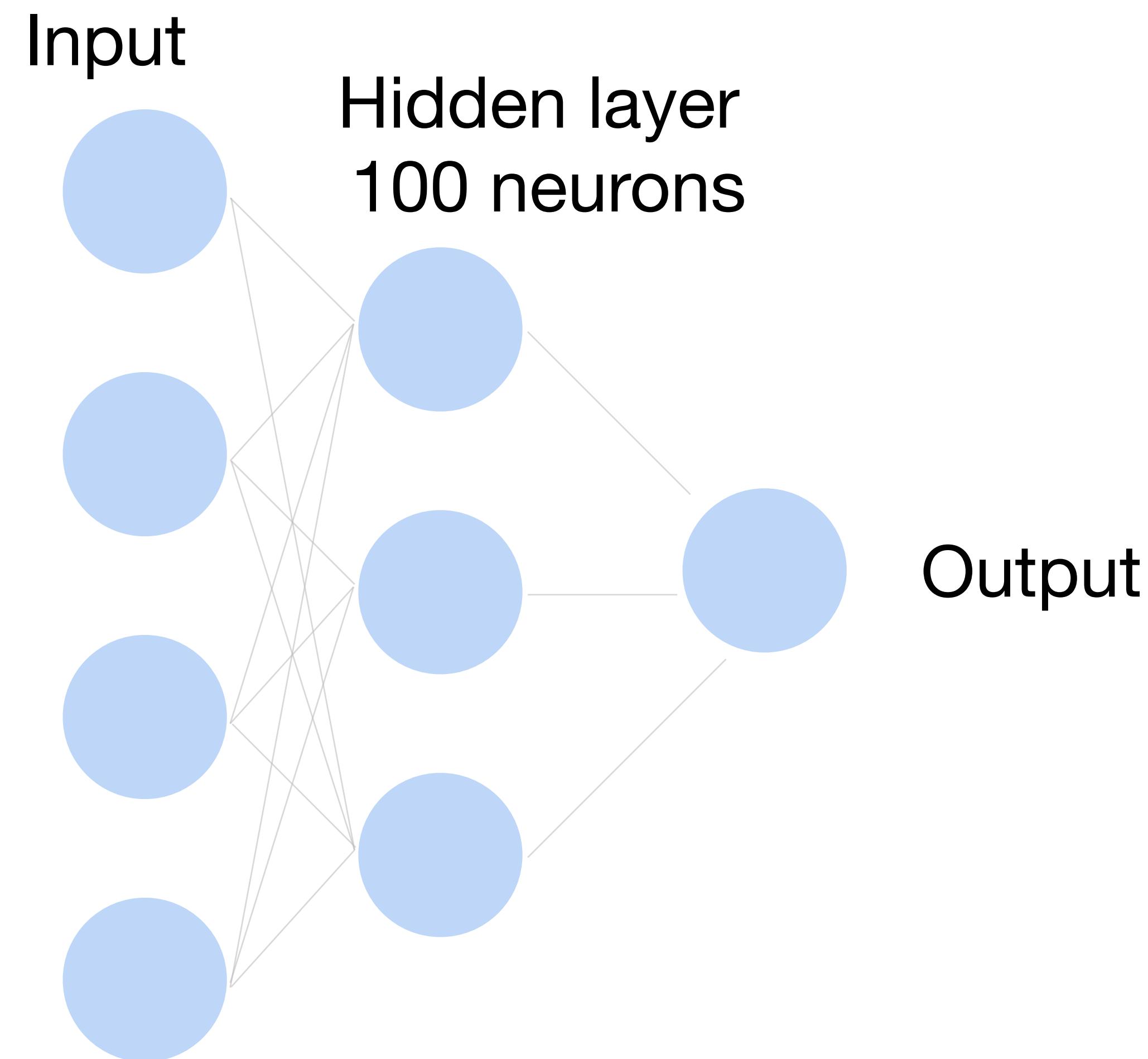
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

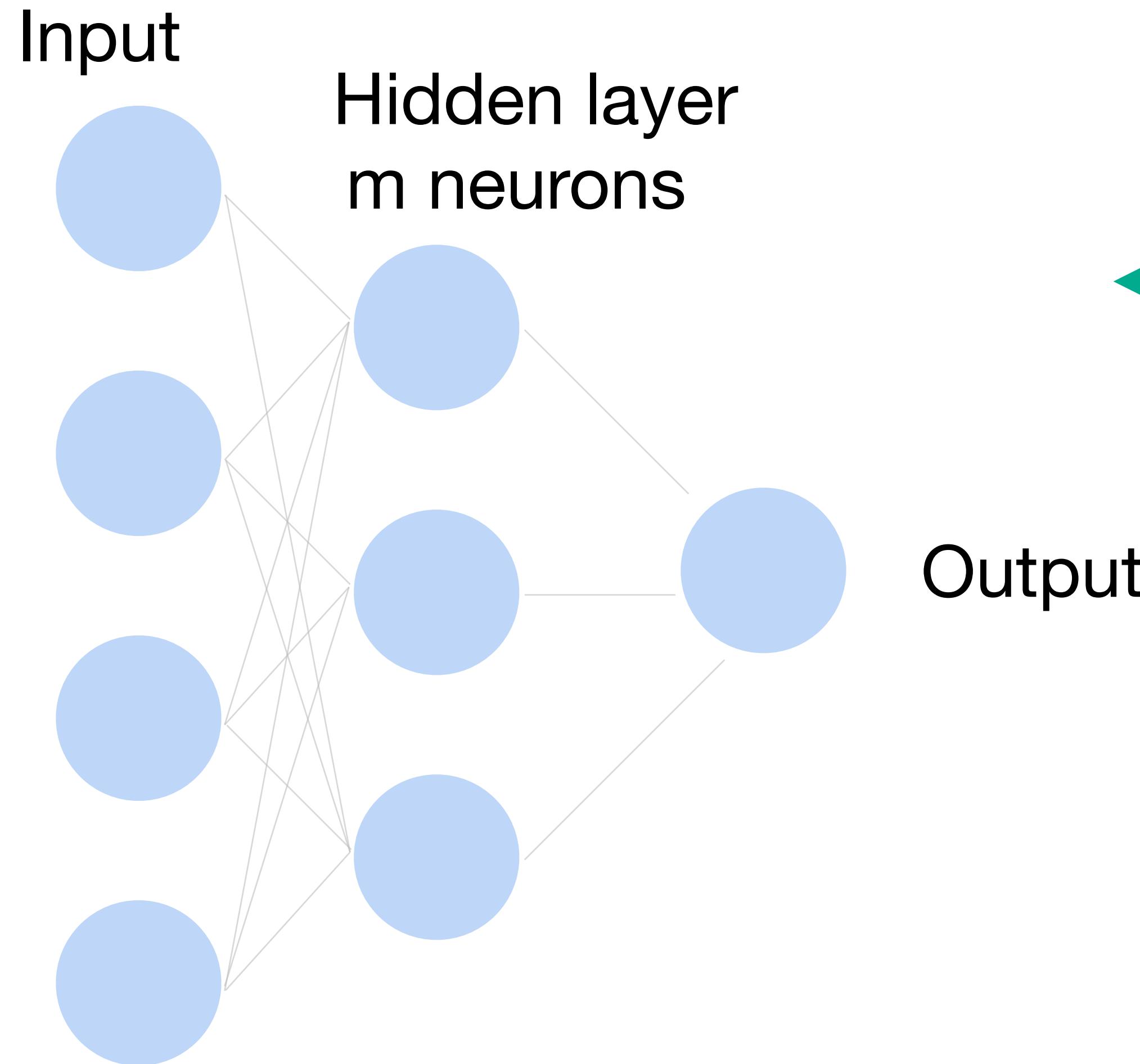
$$\mathbf{h} \in \mathbb{R}^m$$



Classify cats vs. dogs

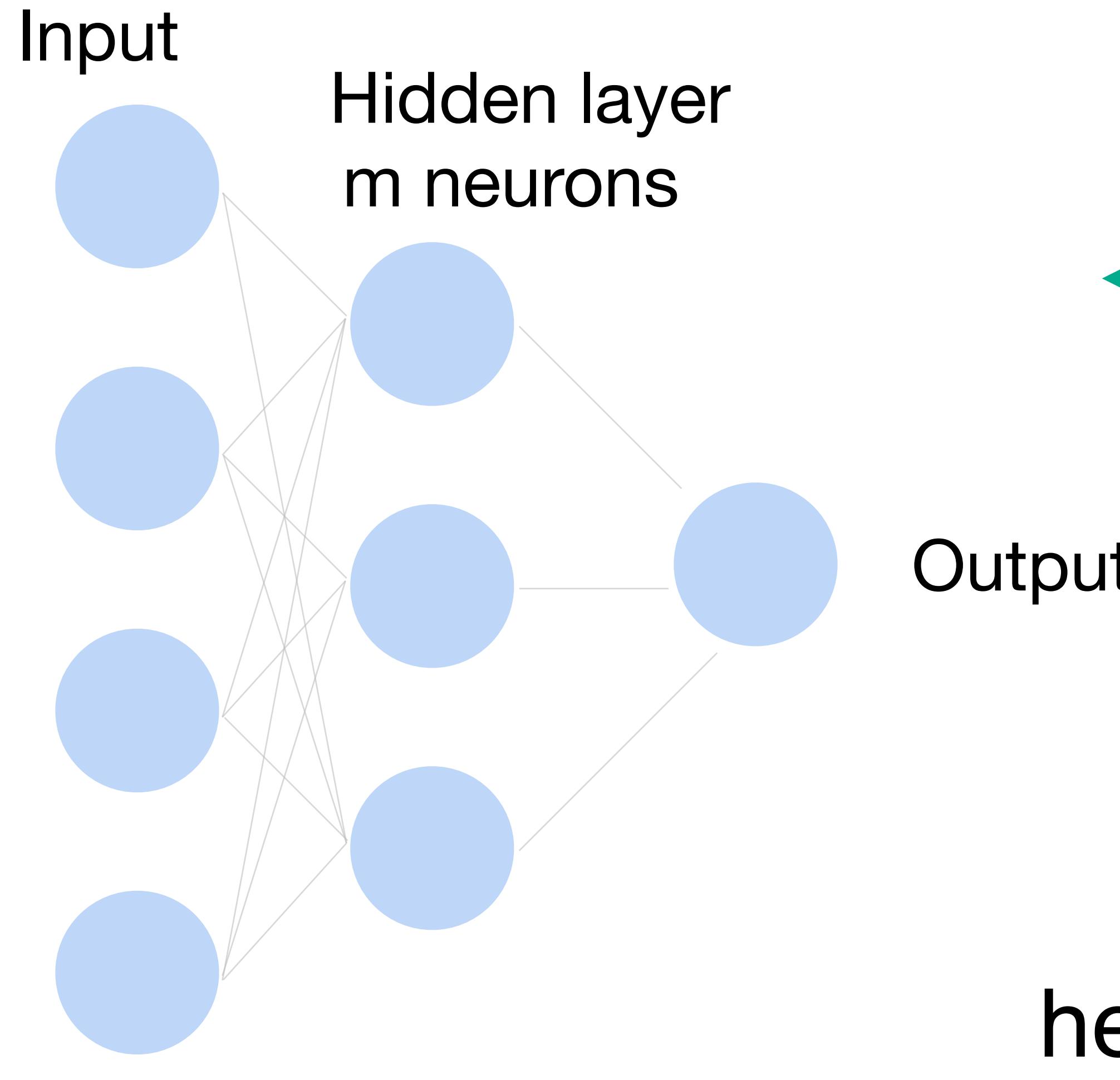


Multi-layer perceptron



Why do we need an a
nonlinear activation?

Multi-layer perceptron



Why do we need an a
nonlinear activation?

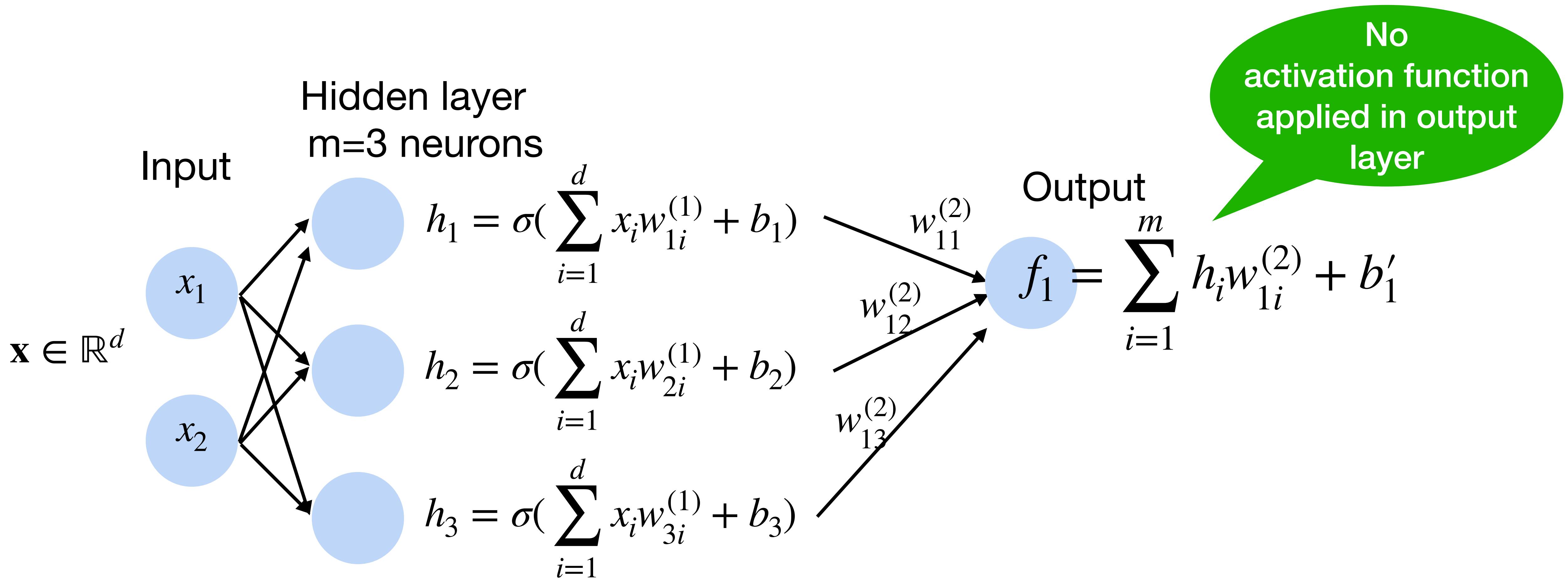
$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$f = \mathbf{w}_2^T \mathbf{h} + b_2$$

$$\text{hence } f = \mathbf{w}_2^T \mathbf{W}\mathbf{x} + b'$$

Neural network for K-way classification

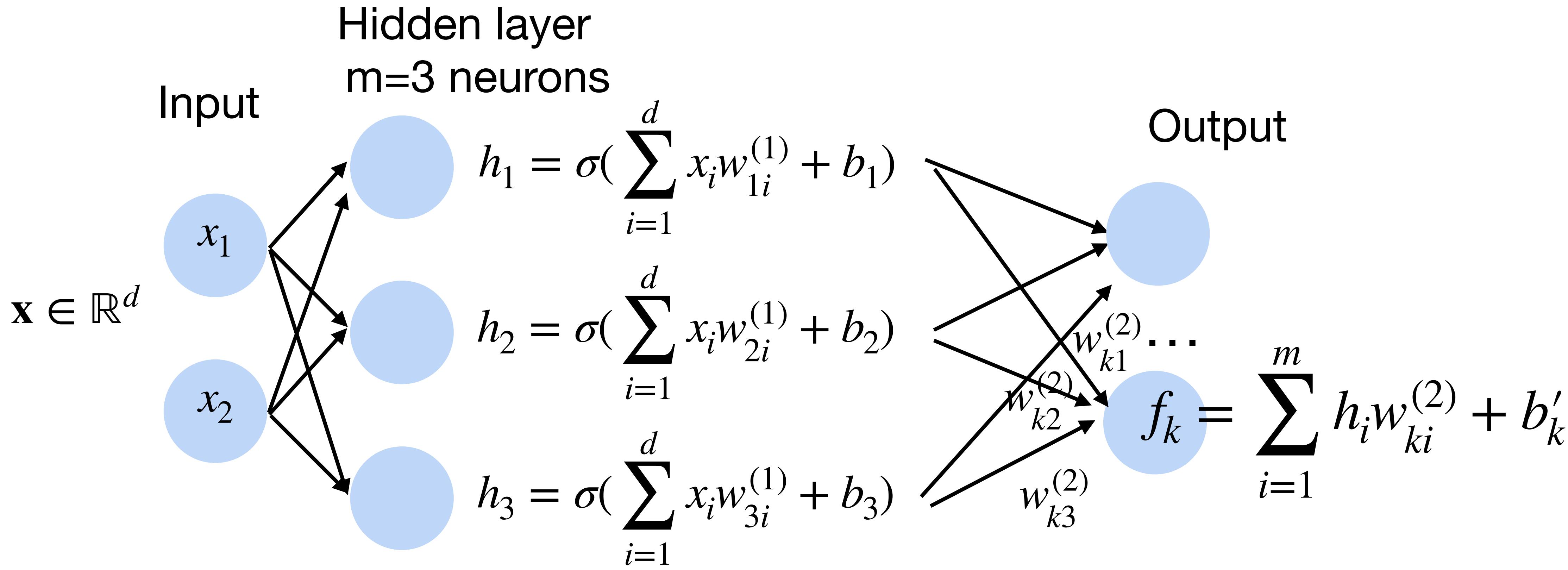
- K outputs in the final layer



Neural network for K-way classification

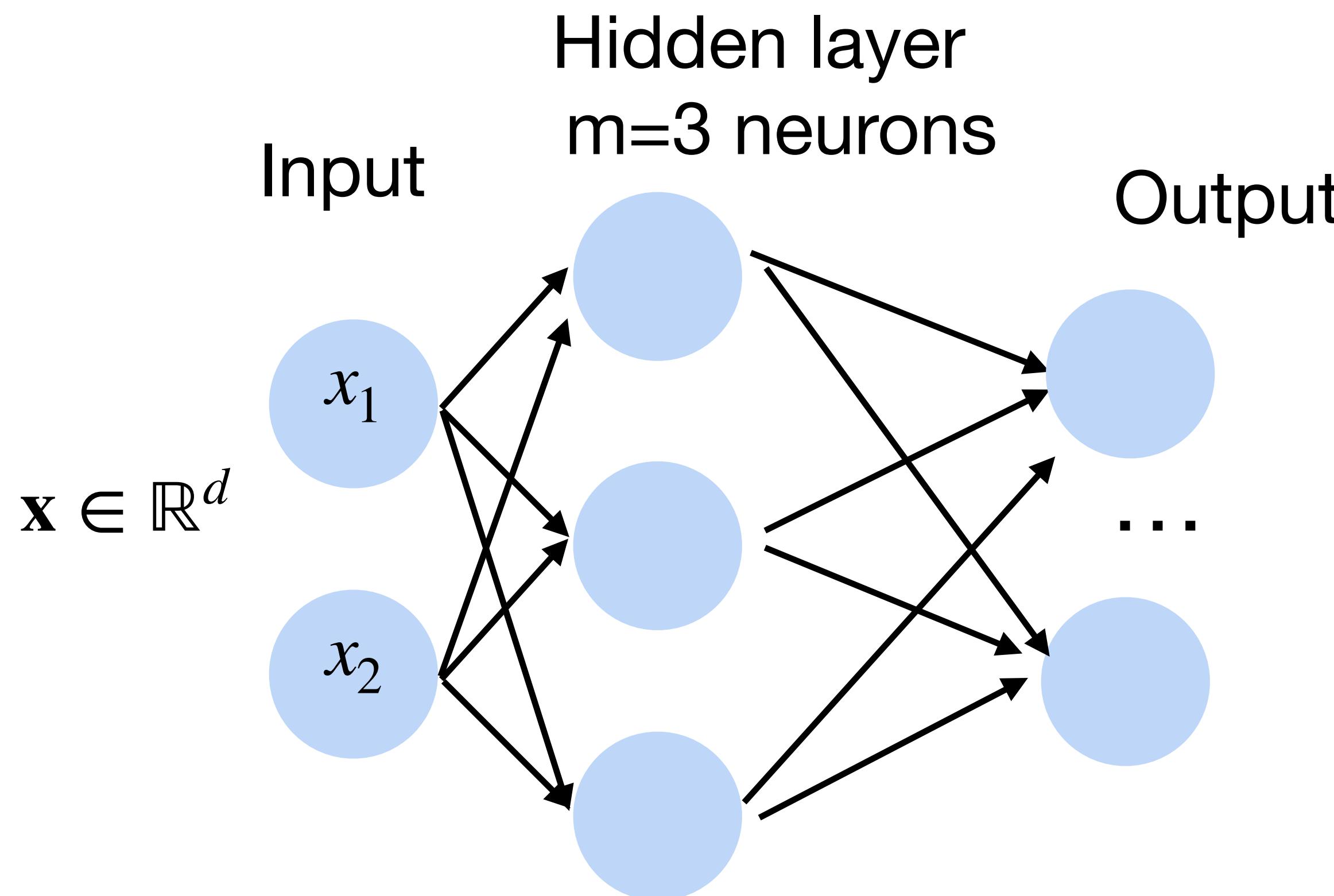
- K outputs units in the final layer

Multi-class classification (e.g., ImageNet with K=1000)



Softmax

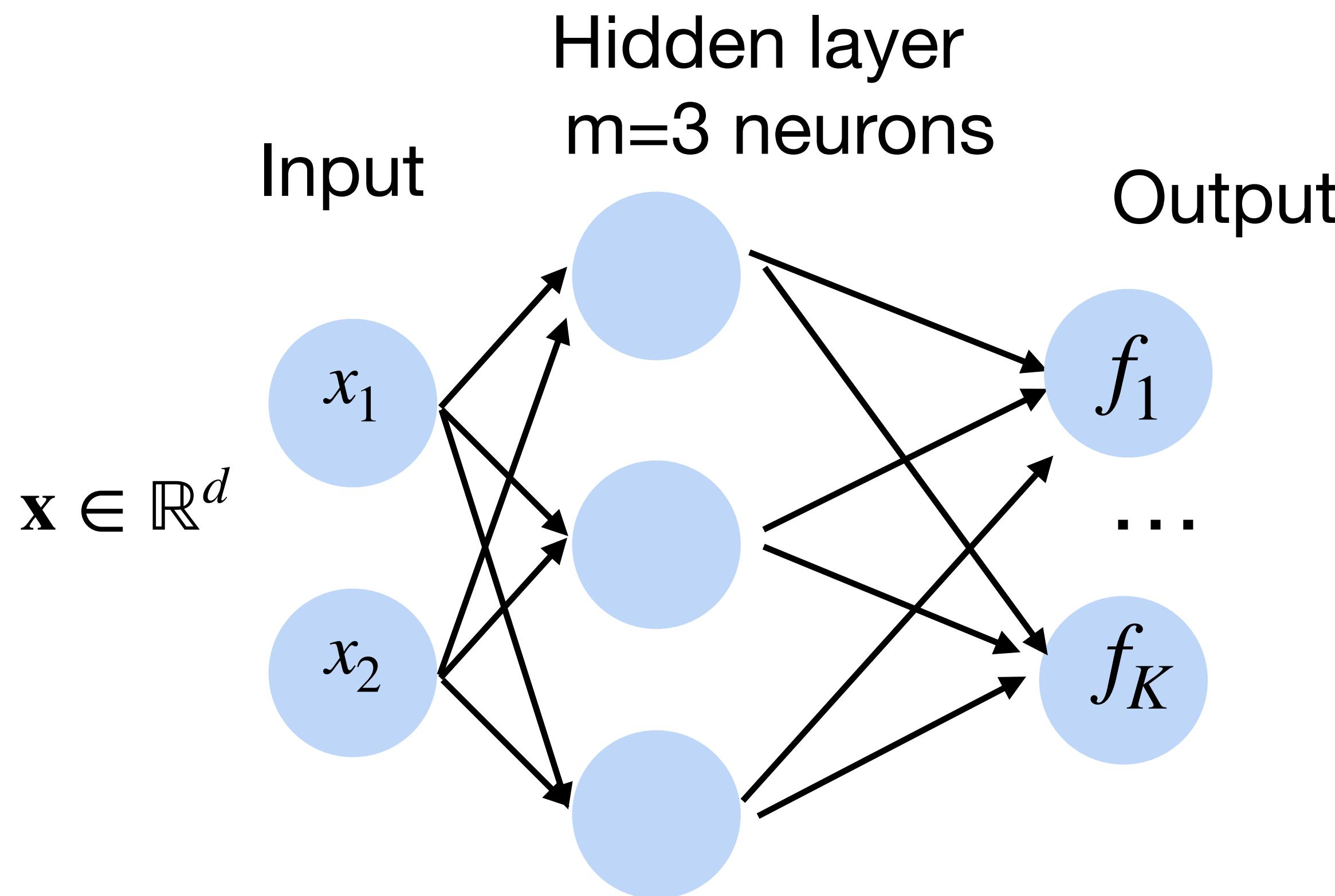
Turns outputs f into probabilities (sum up to 1 across K classes)



$$p(y | \mathbf{x}) = \text{softmax}(f)$$
$$= \frac{\exp f_y(x)}{\sum_{k=1}^K \exp f_k(x)}$$

Softmax

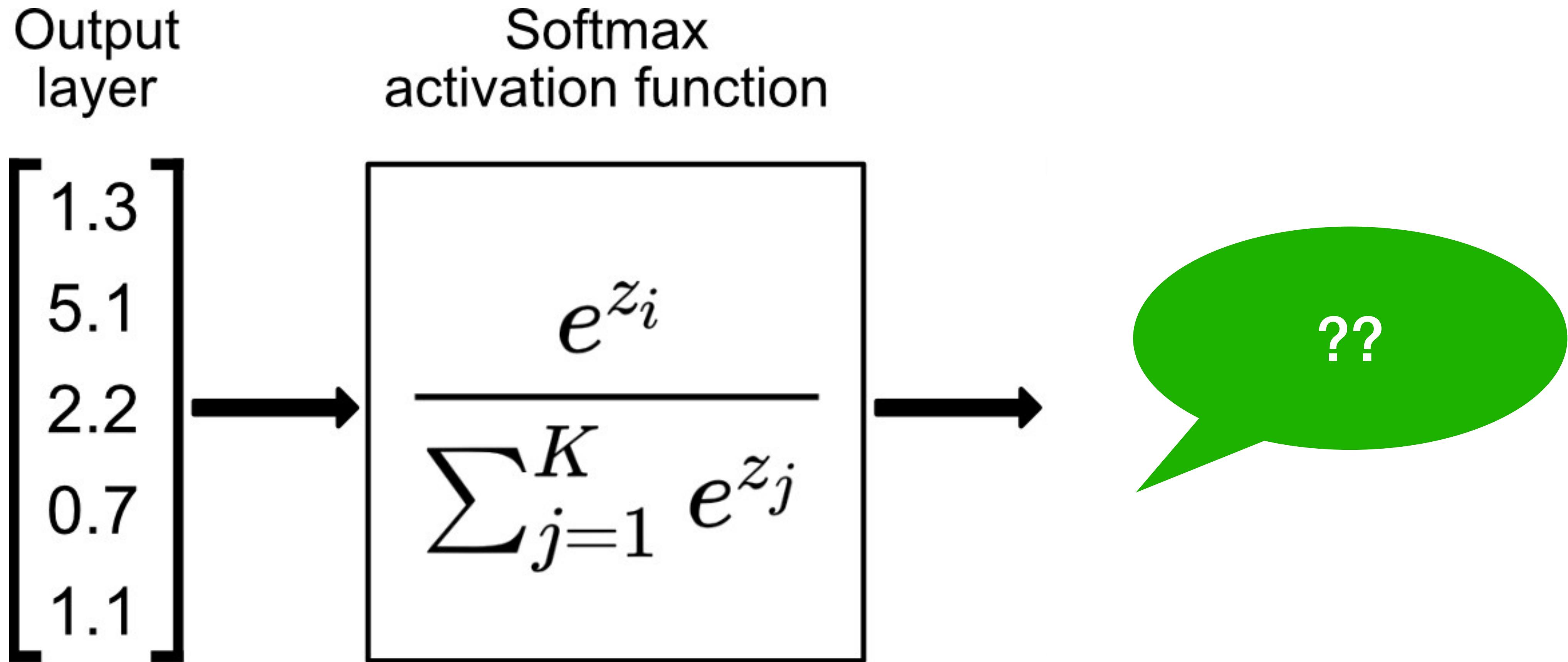
Turns outputs f into probabilities (sum up to 1 across K classes)



$$p(y | \mathbf{x}) = \text{softmax}(f) = \frac{\exp f_y(x)}{\sum_{k=1}^K \exp f_k(x)}$$

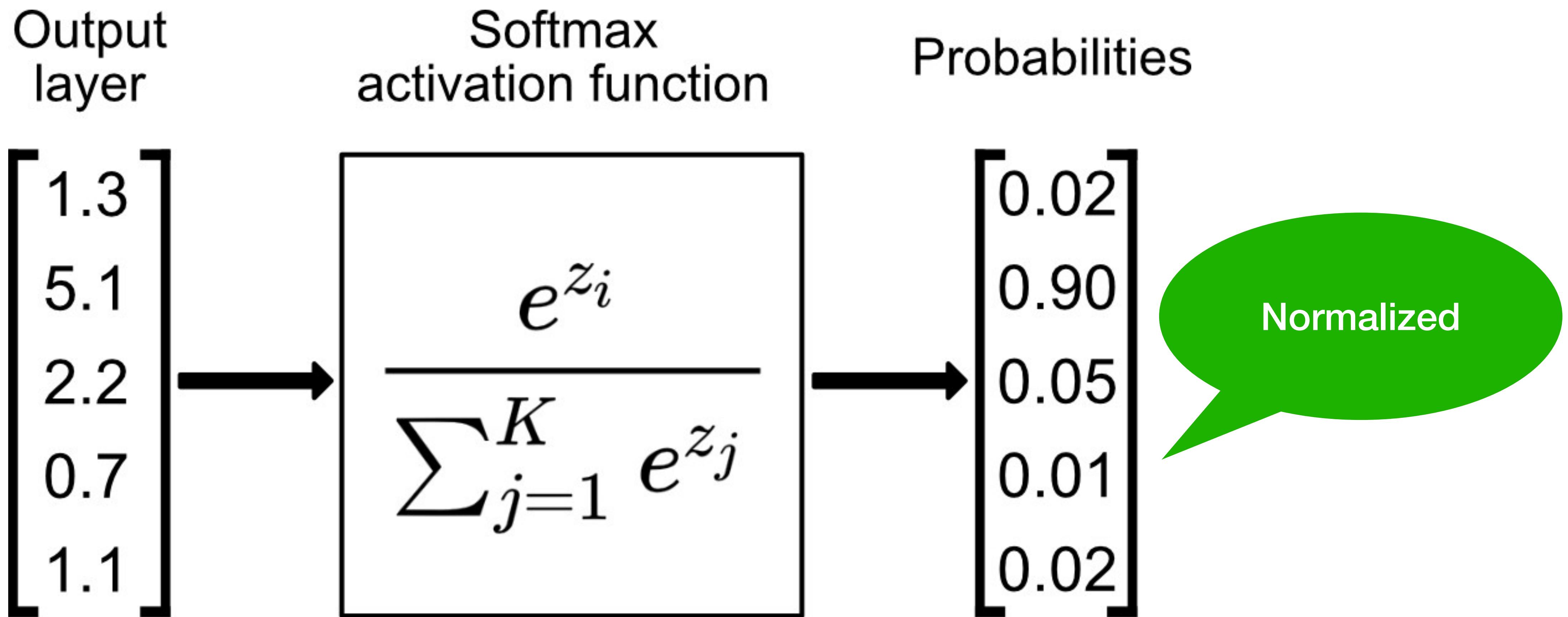
Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)



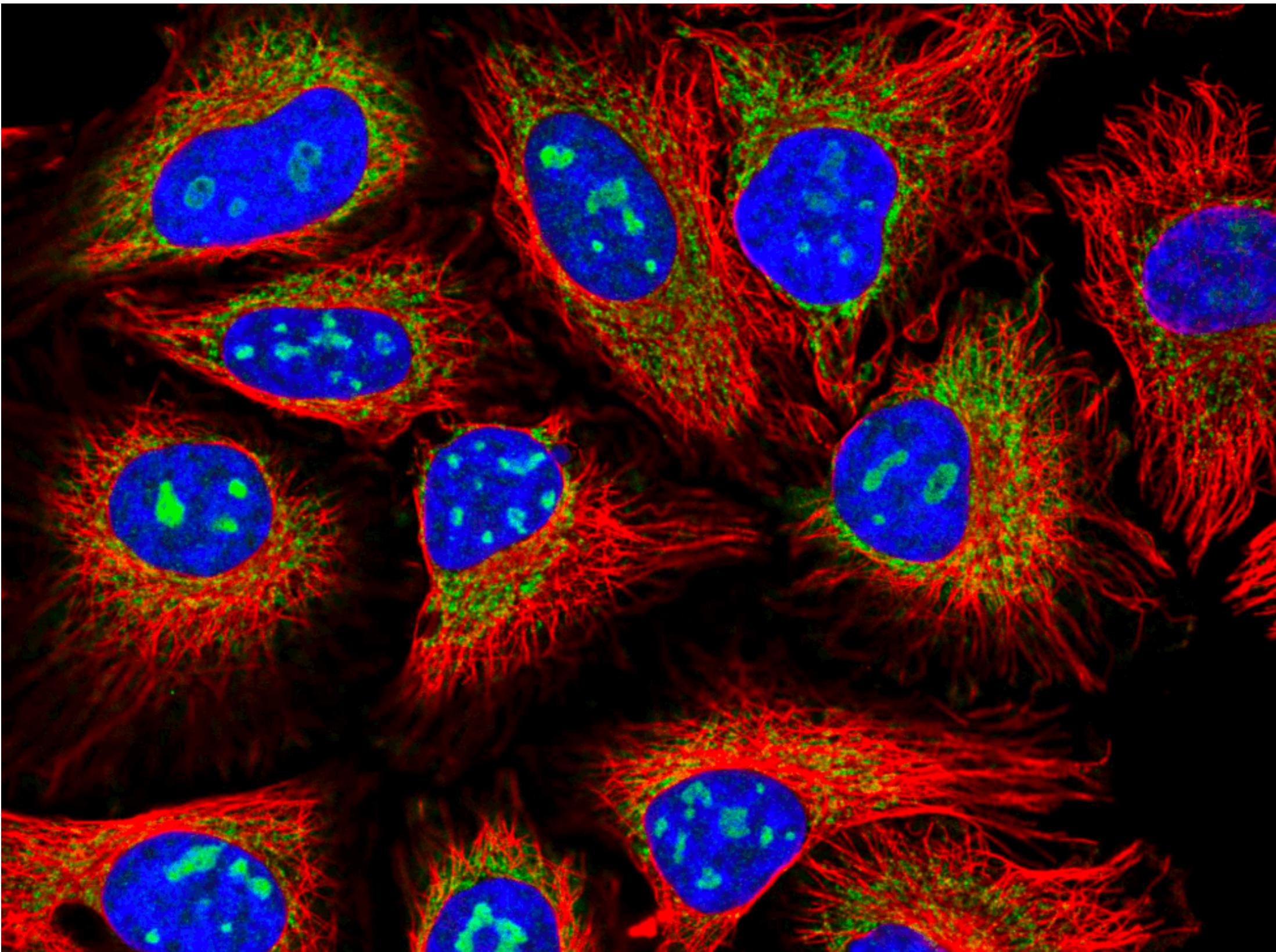
Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)



Classification Tasks at Kaggle

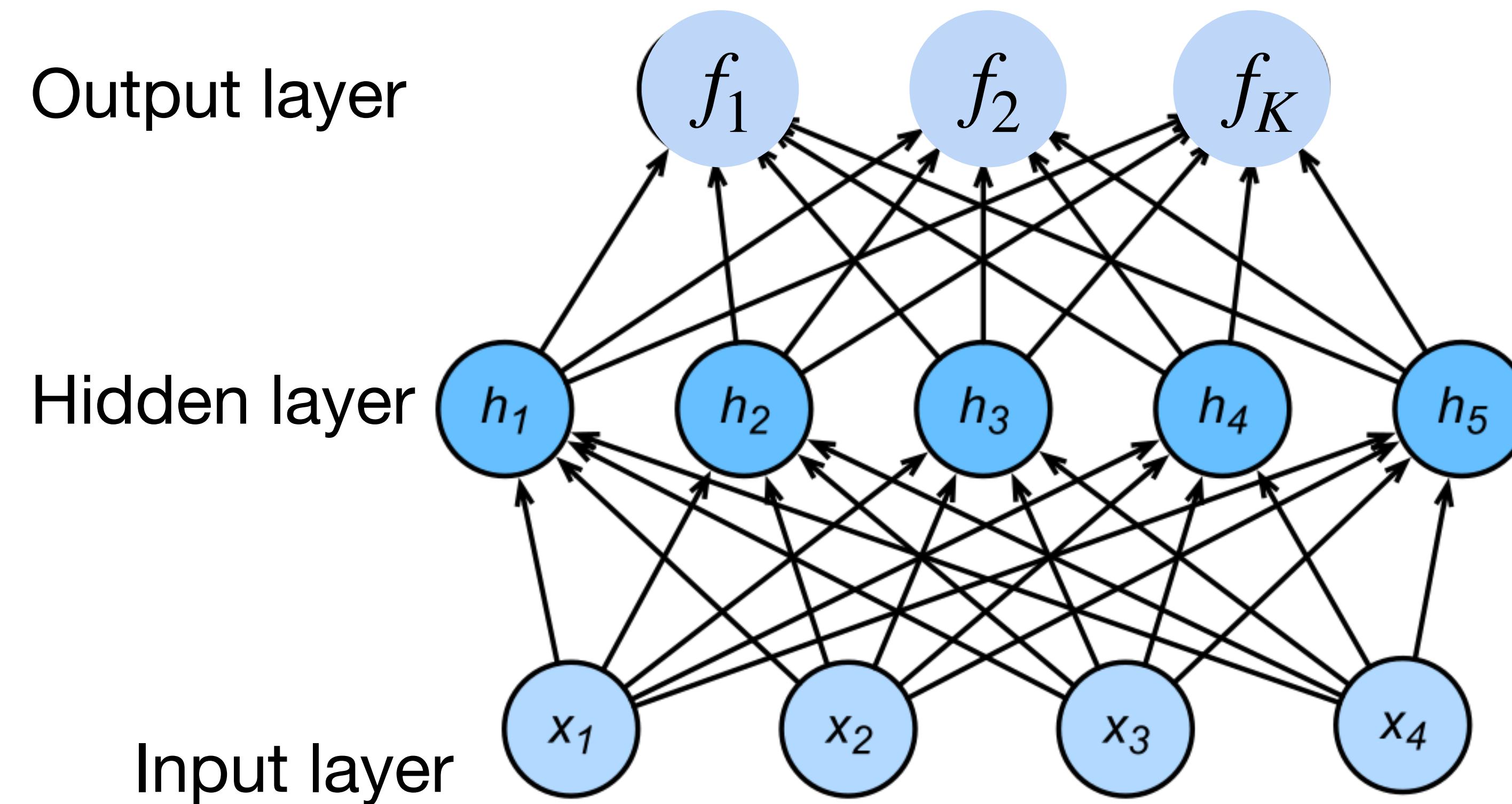
Classify human protein microscope images into 28 categories



- 0. Nucleoplasm
- 1. Nuclear membrane
- 2. Nucleoli
- 3. Nucleoli fibrillar
- 4. Nuclear speckles
- 5. Nuclear bodies
- 6. Endoplasmic reticu
- 7. Golgi apparatus
- 8. Peroxisomes
- 9. Endosomes
- 10. Lysosomes
- 11. Intermediate fila
- 12. Actin filaments
- 13. Focal adhesion si
- 14. Microtubules
- 15. Microtubule ends
- 16. Cytokinetic brida

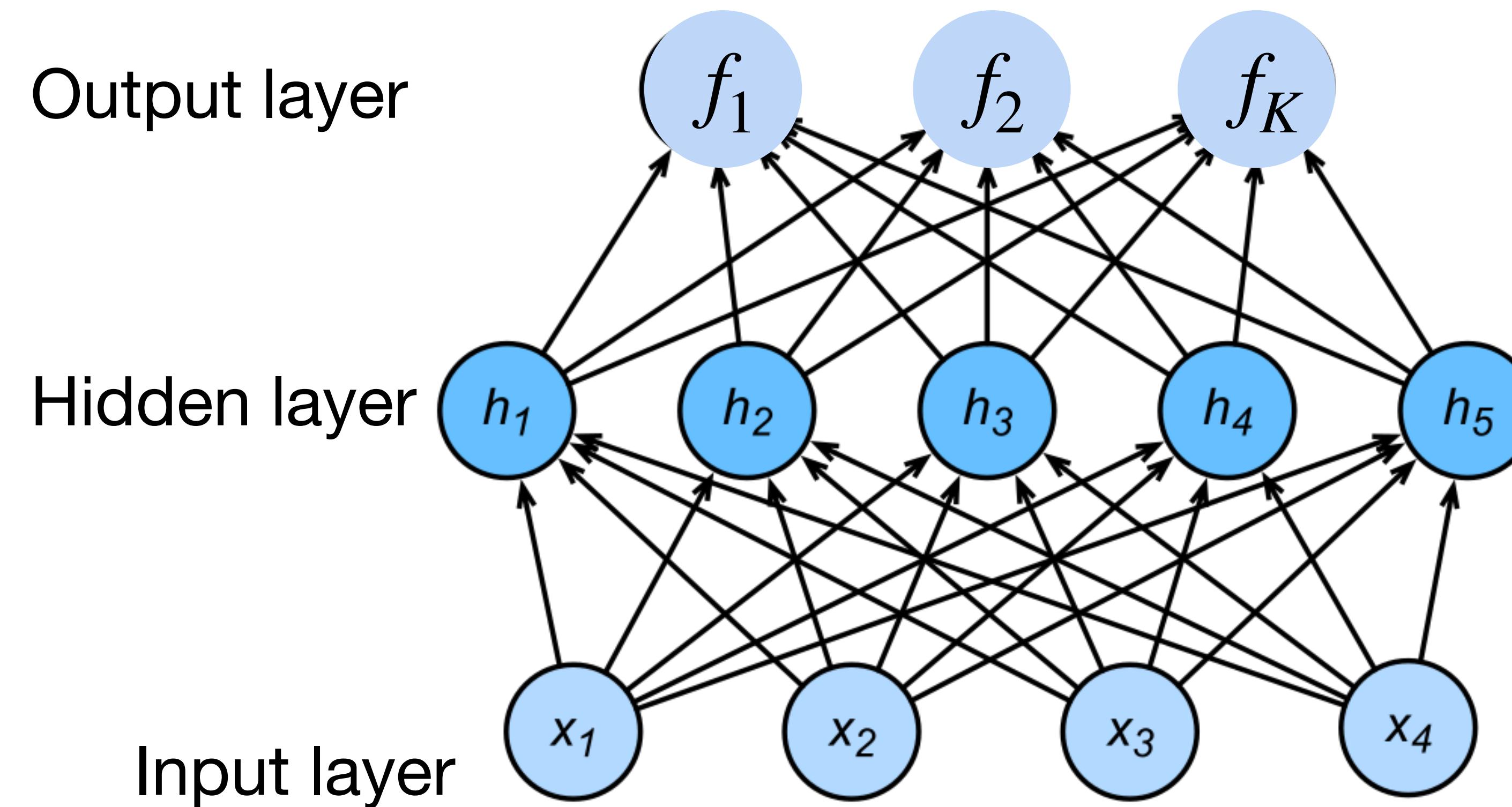
<https://www.kaggle.com/c/human-protein-atlas-image-classification>

More complicated neural networks



More complicated neural networks

$$p_1, p_2, \dots, p_K = \text{softmax}(f_1, f_2, \dots, f_K)$$



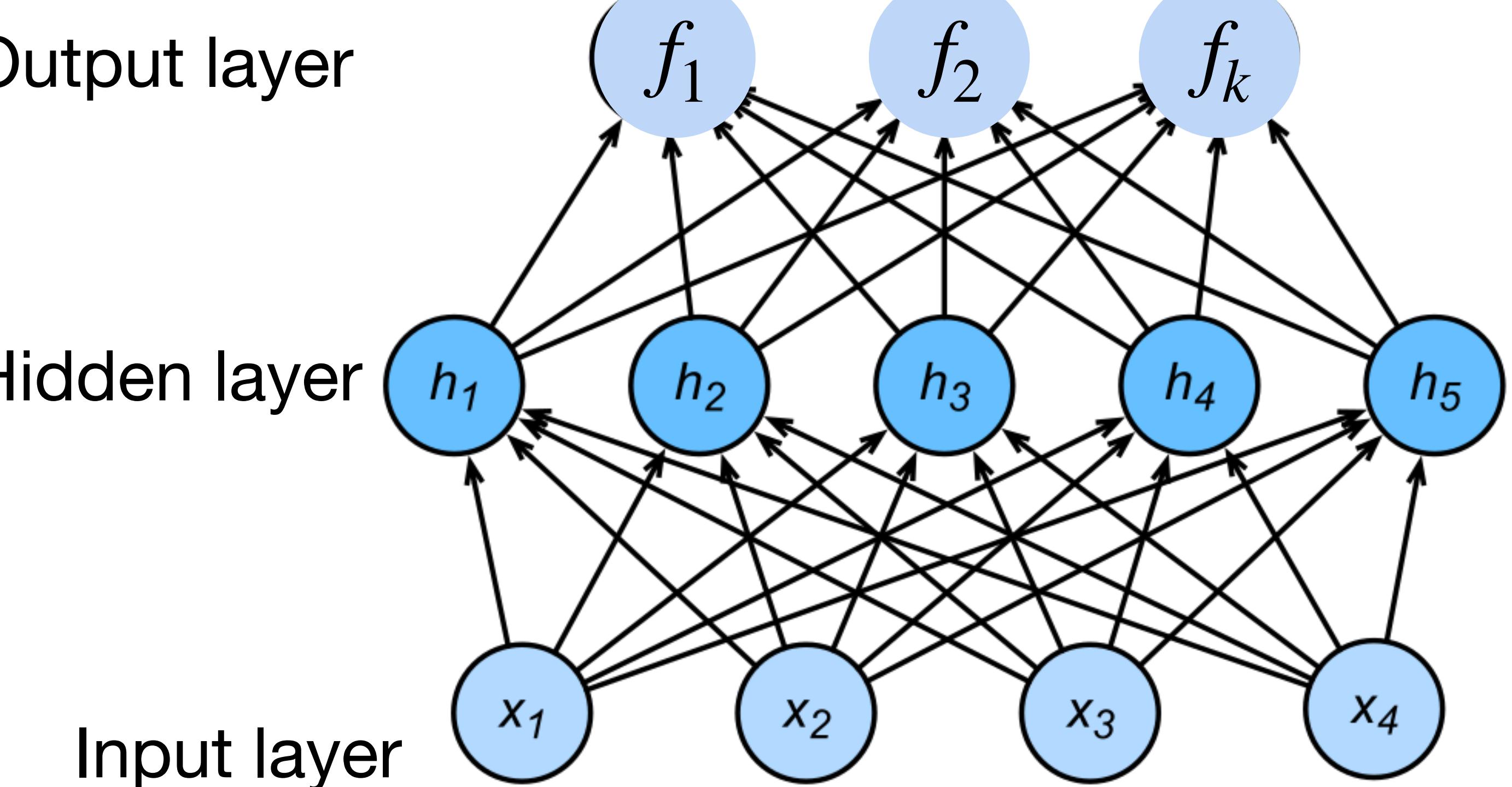
More complicated neural networks

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



More complicated neural networks

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

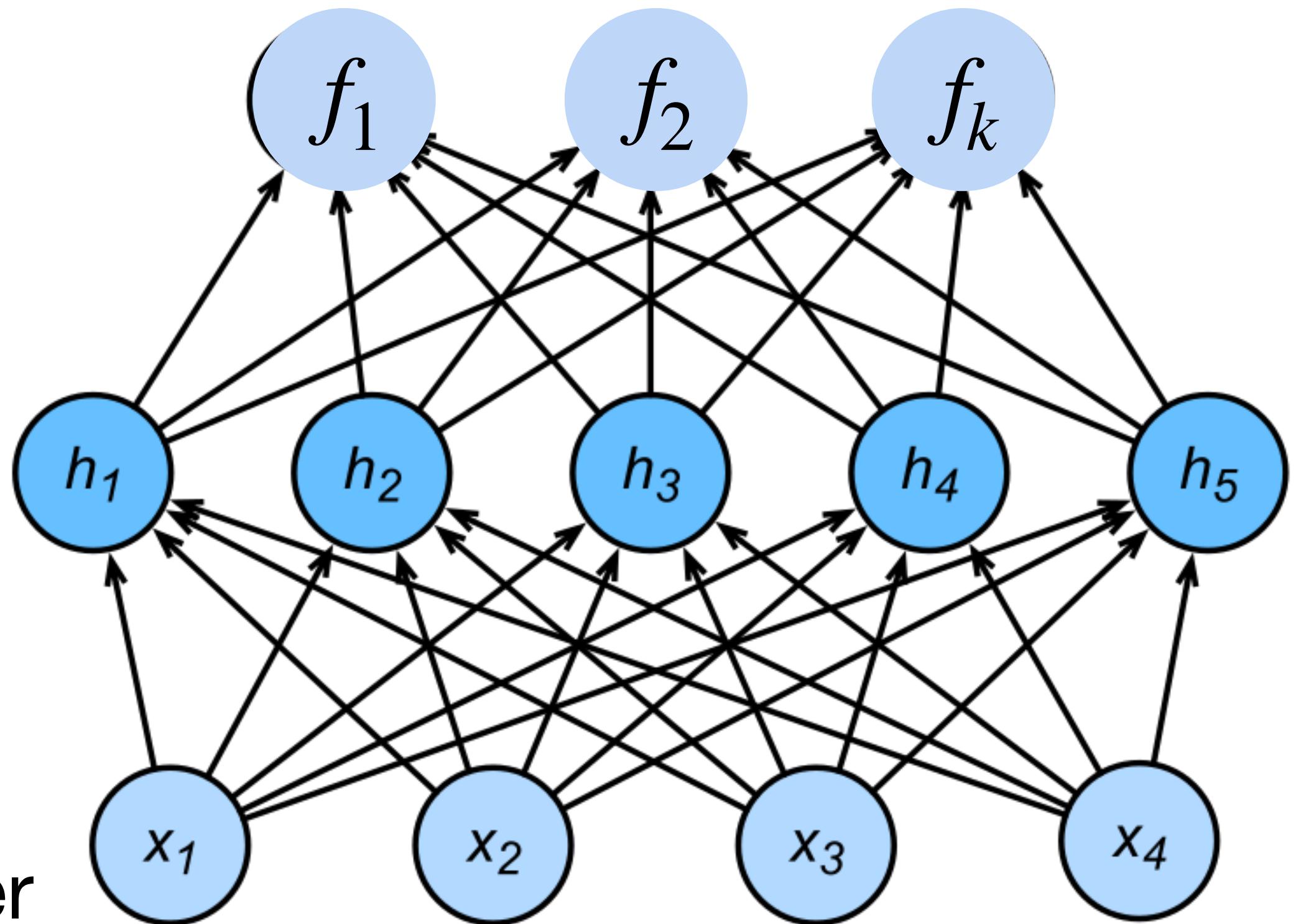
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$

$$p_1, p_2, \dots, p_K = \text{softmax}(f_1, f_2, \dots, f_K)$$

Output layer

Hidden layer

Input layer



More complicated neural networks: multiple hidden layers

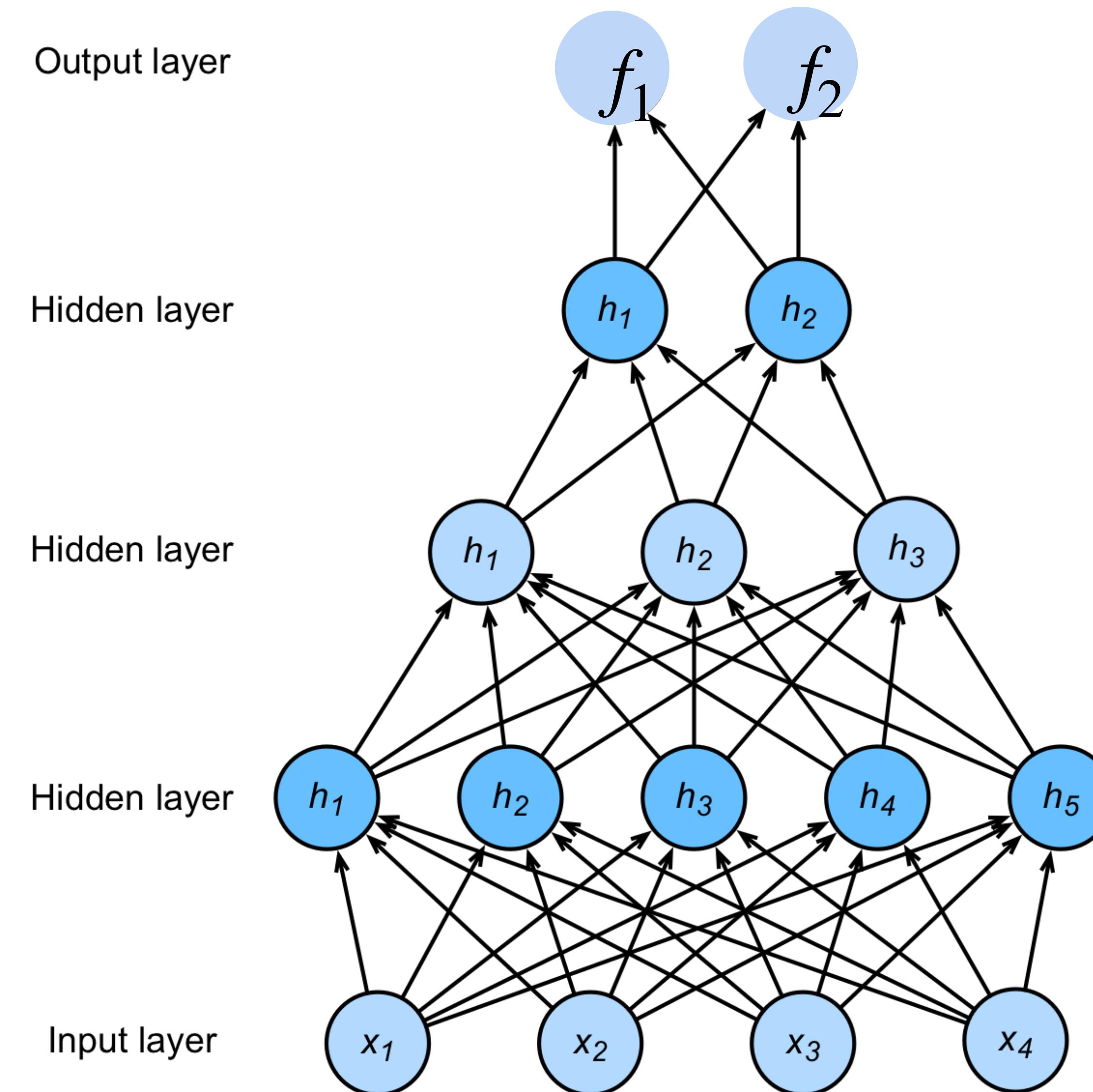
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

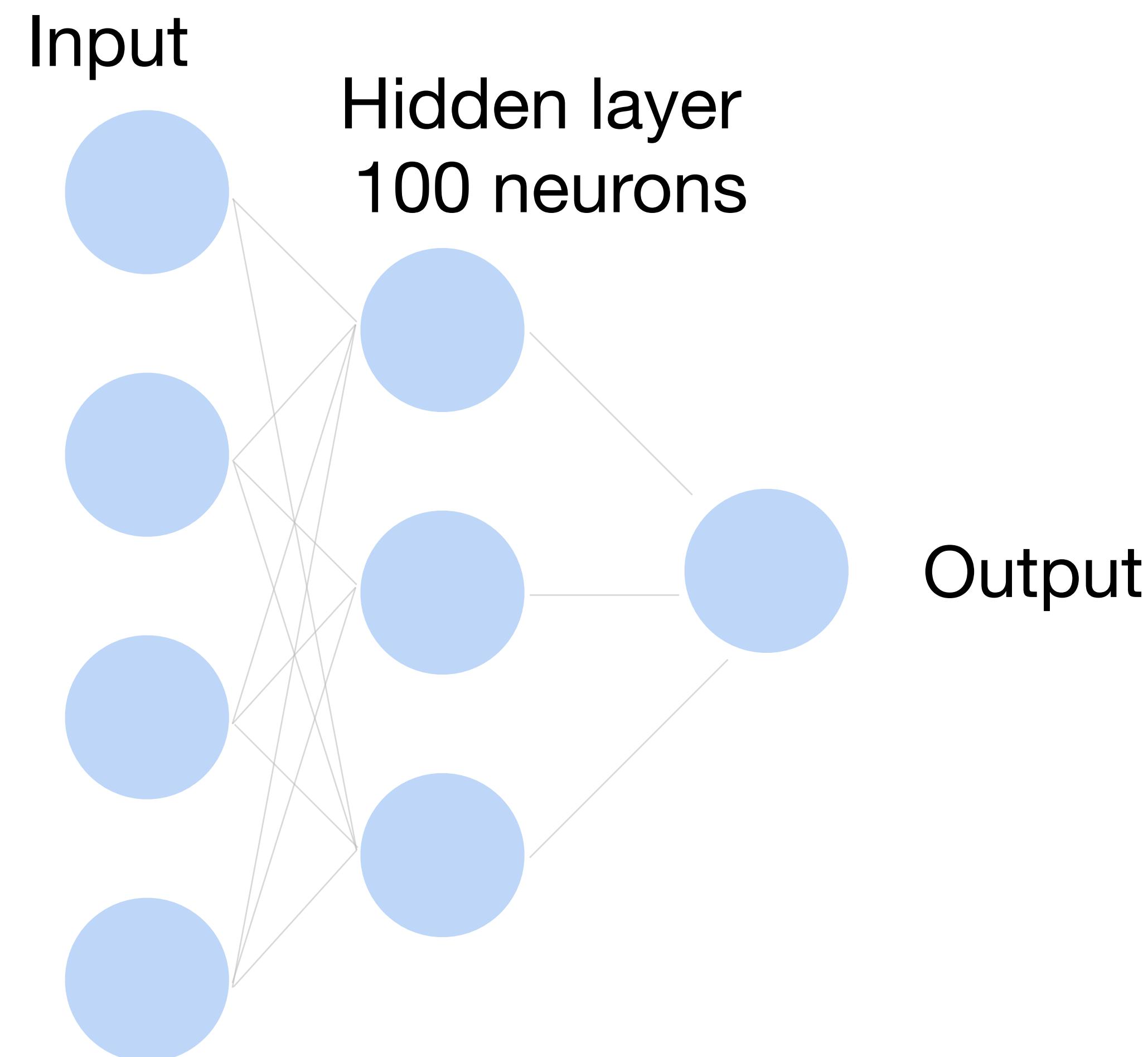
$$\mathbf{f} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



How to train a neural network?

Classify cats vs. dogs



How to train a neural network? Binary classification

$\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

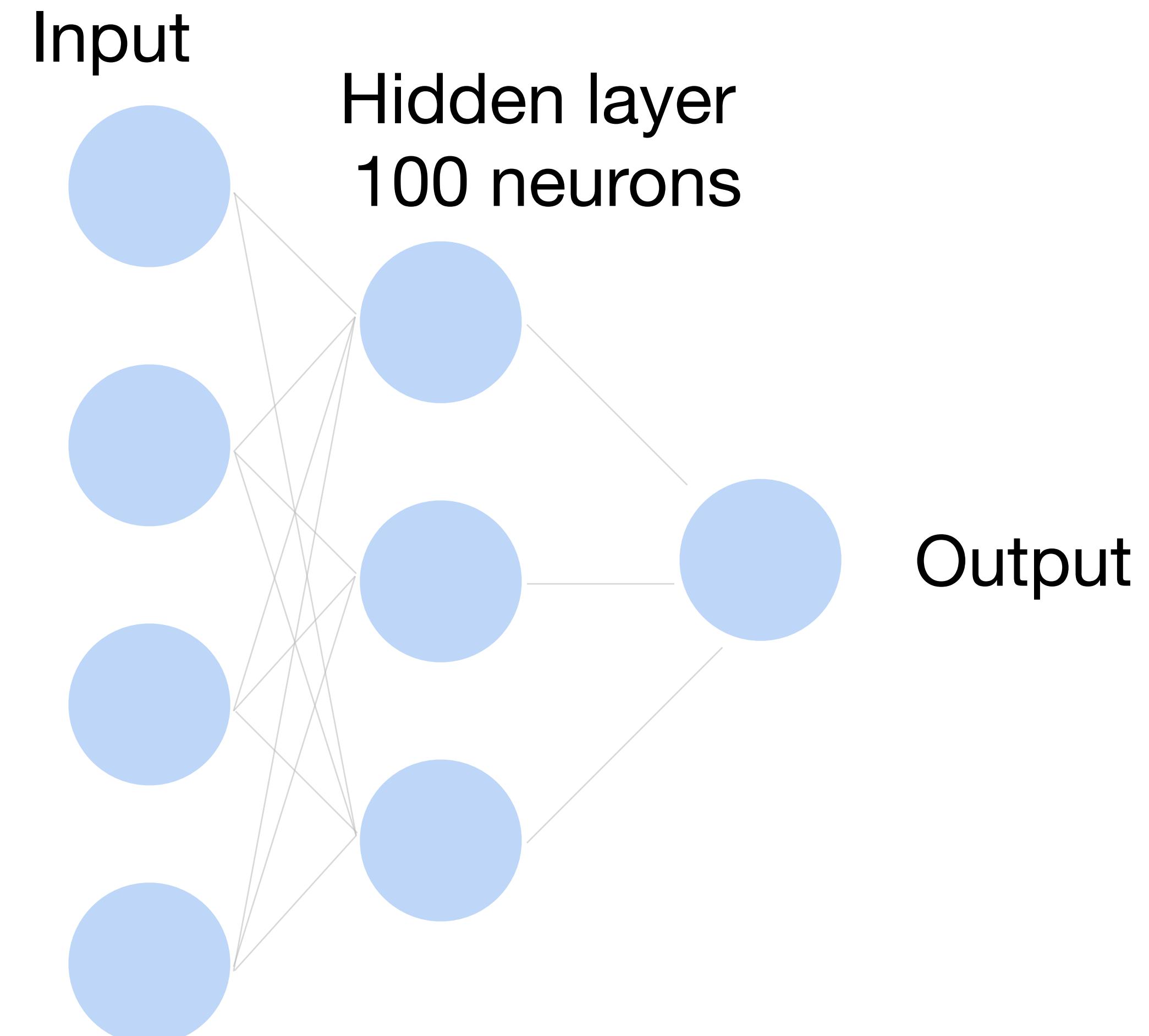
\hat{y} Model output for example \mathbf{x}

(This is a function of all weights W)

y Ground truth label for example \mathbf{x}

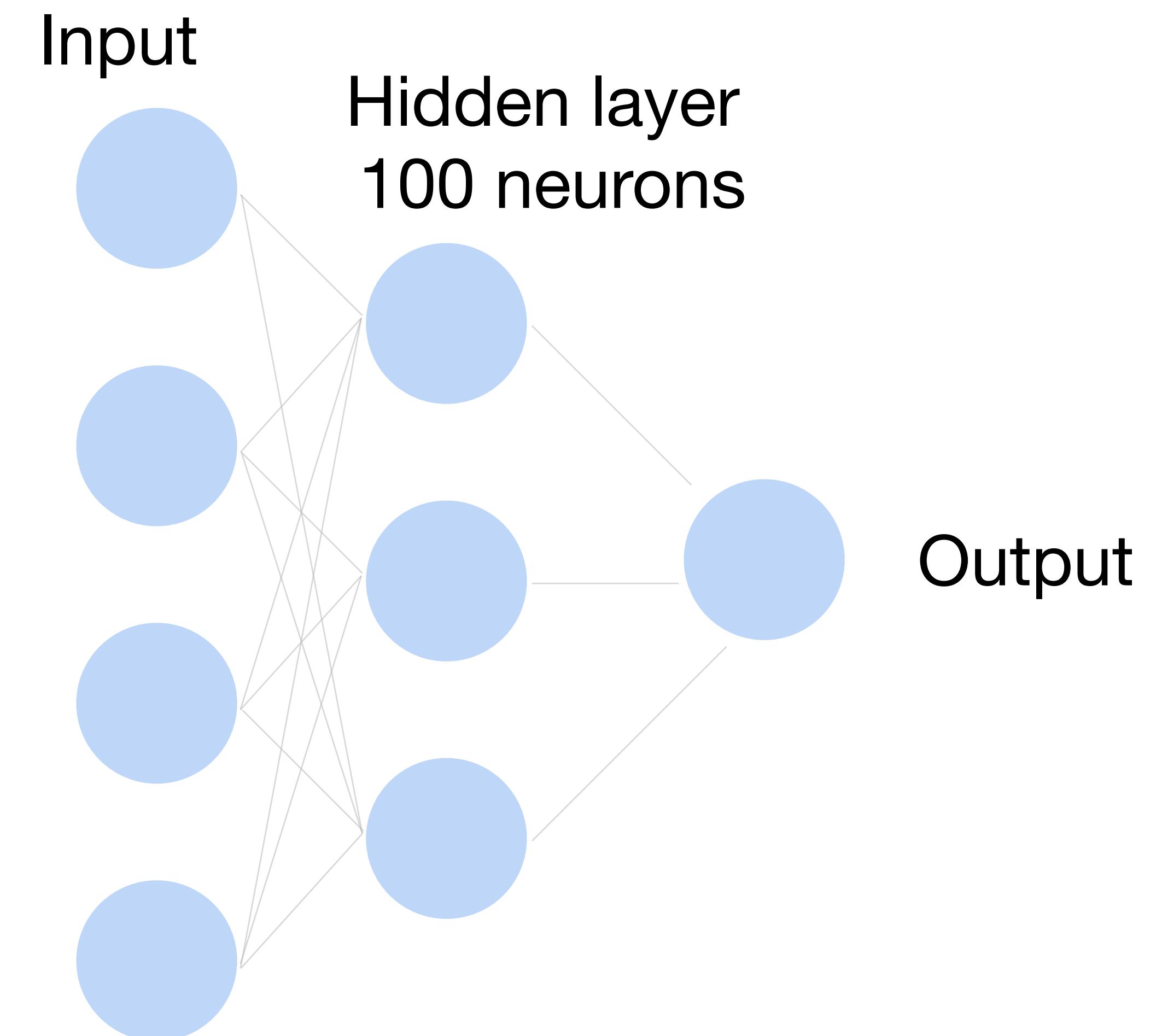
**Learning by matching the output
to the label**

We want $\hat{y} \rightarrow 1$ when $y = 1$,
and $\hat{y} \rightarrow 0$ when $y = 0$



How to train a neural network? Binary classification

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

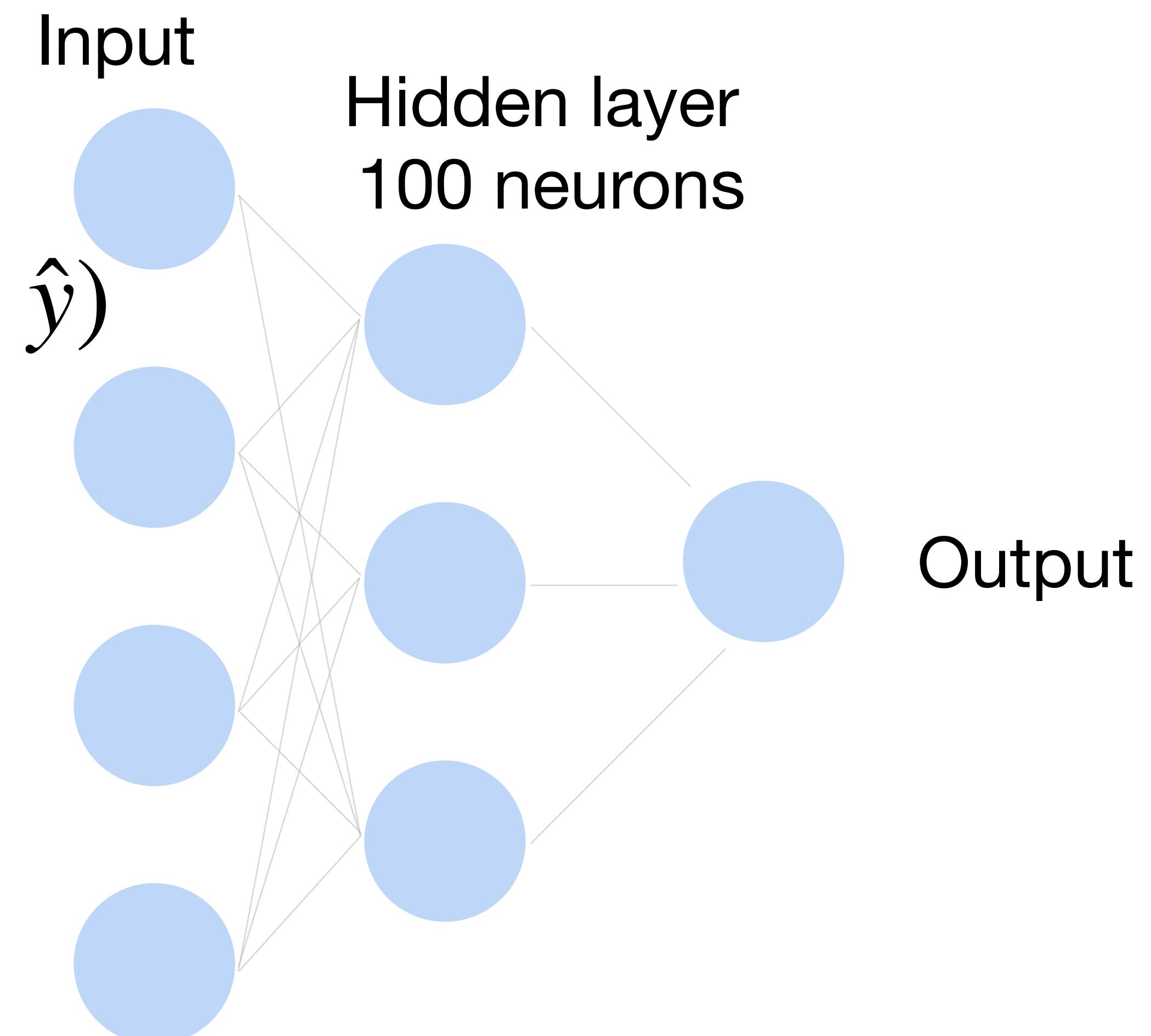


How to train a neural network? Binary classification

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

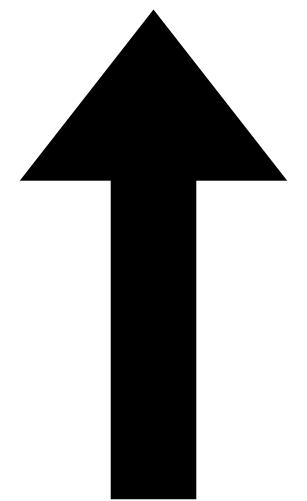


How to train a neural network? Binary classification

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

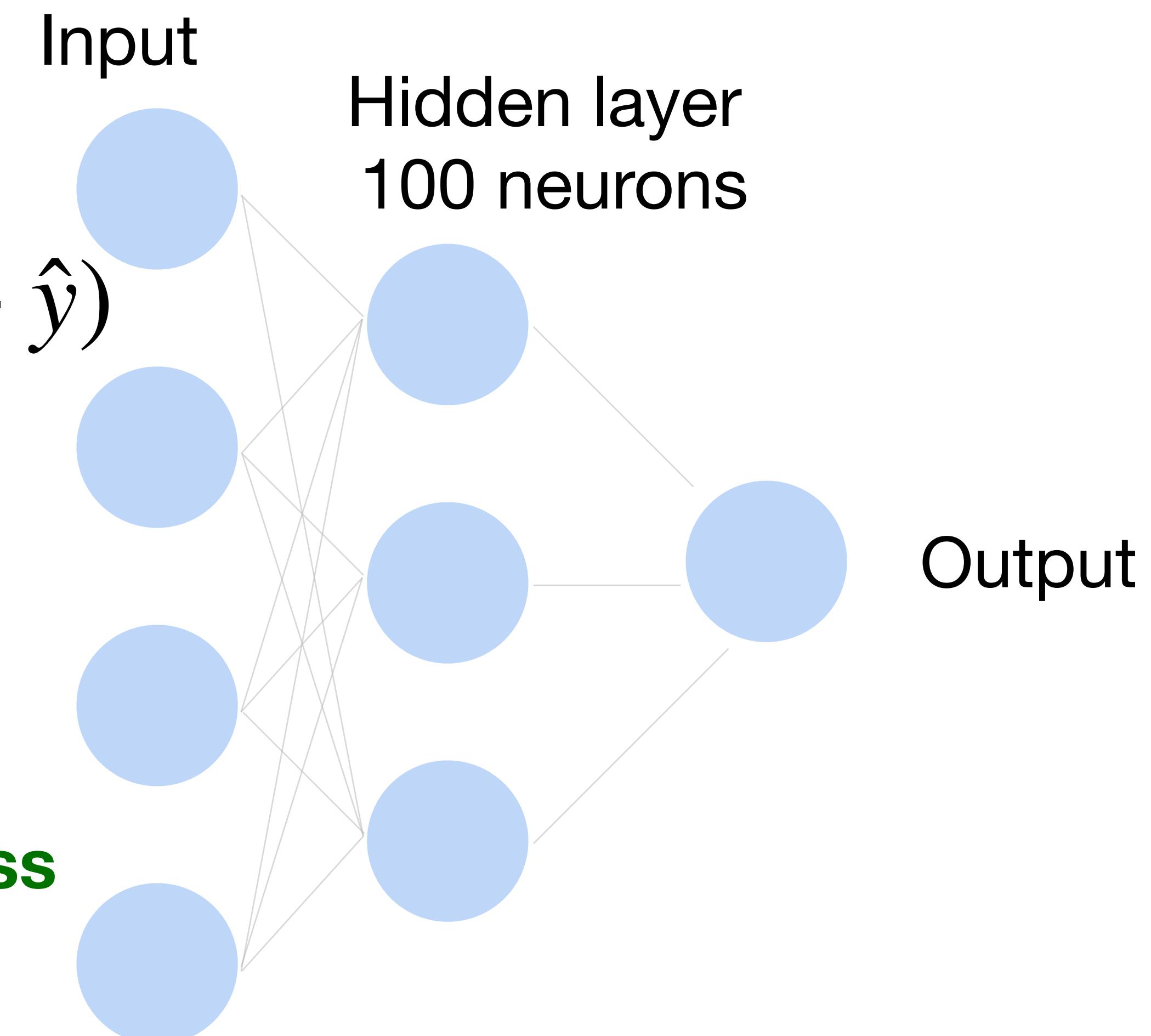
Per-sample loss:

$$\ell(\mathbf{x}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$



Negative log likelihood

Also known as **binary cross-entropy loss**

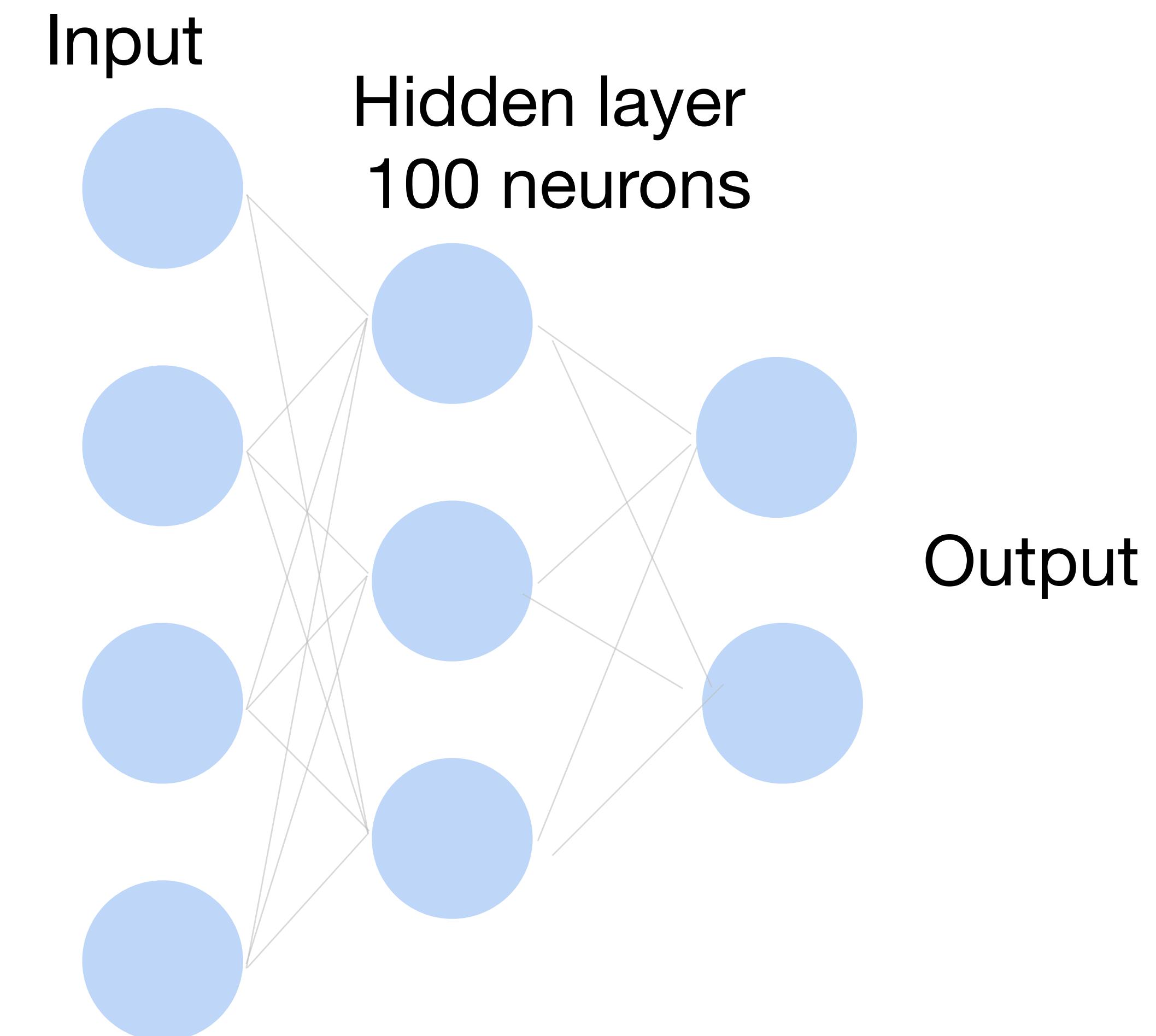


How to train a neural network? Multiclass

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Y

y



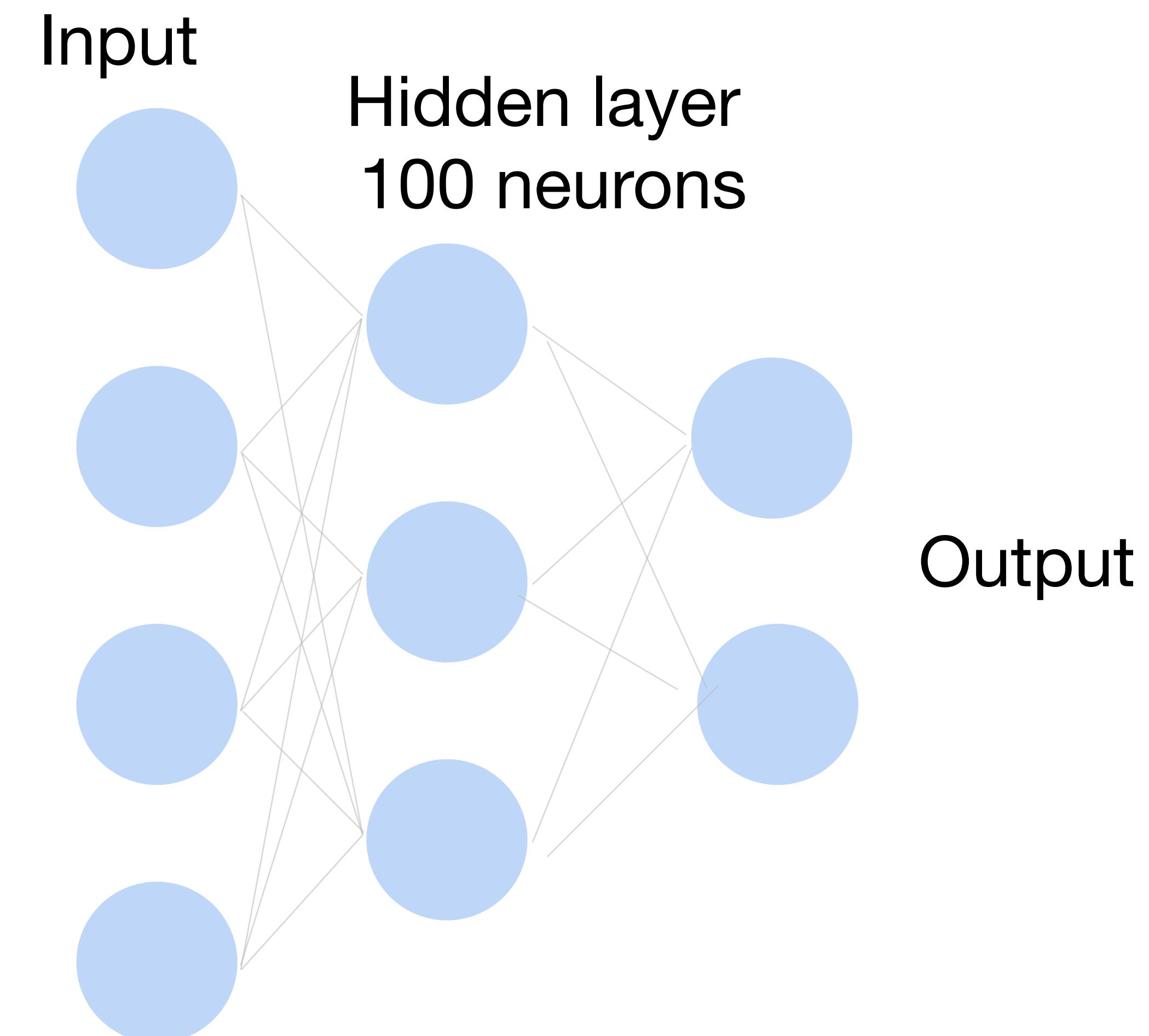
How to train a neural network? Multiclass

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{k=1}^K -Y_k \log p_k = -\log p_y$$

where Y is one-hot encoding of y



How to train a neural network? Multiclass

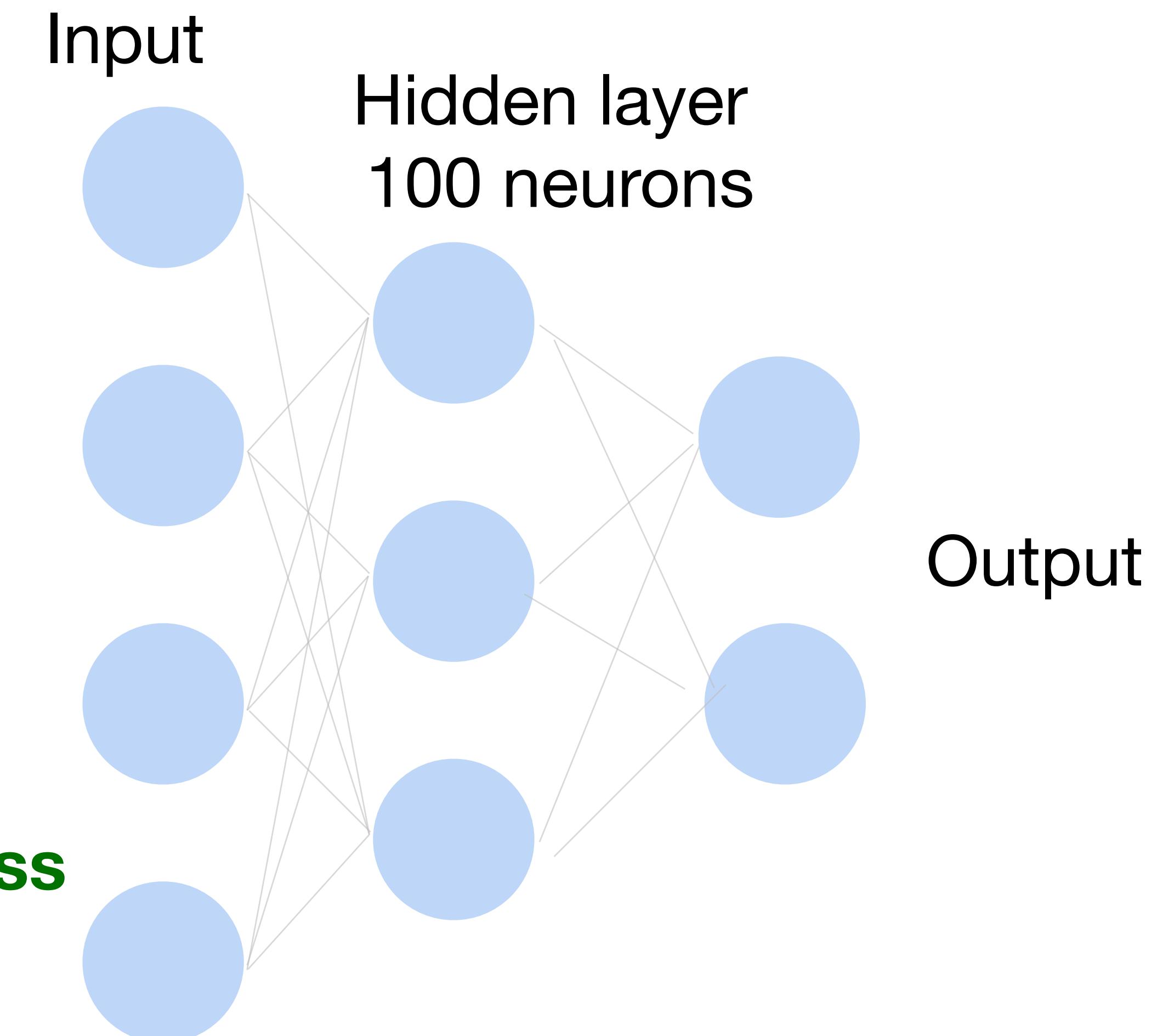
Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{k=1}^K -Y_k \log p_k = -\log p_y$$

where Y is one-hot encoding of y

Also known as **cross-entropy loss**
or **softmax loss**

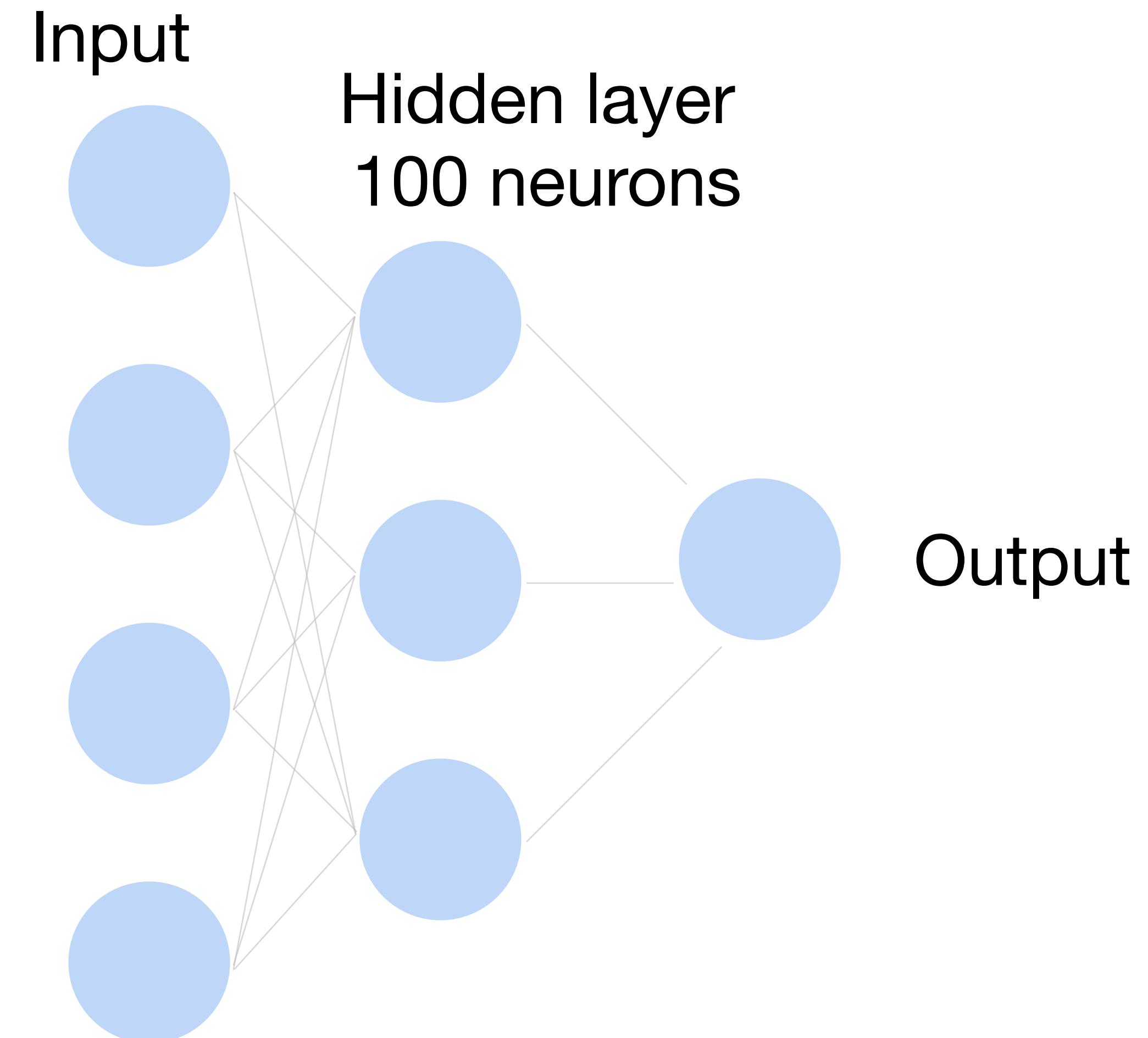


How to train a neural network?

Update the weights W to minimize the loss function

$$L = \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$$

Use gradient descent!



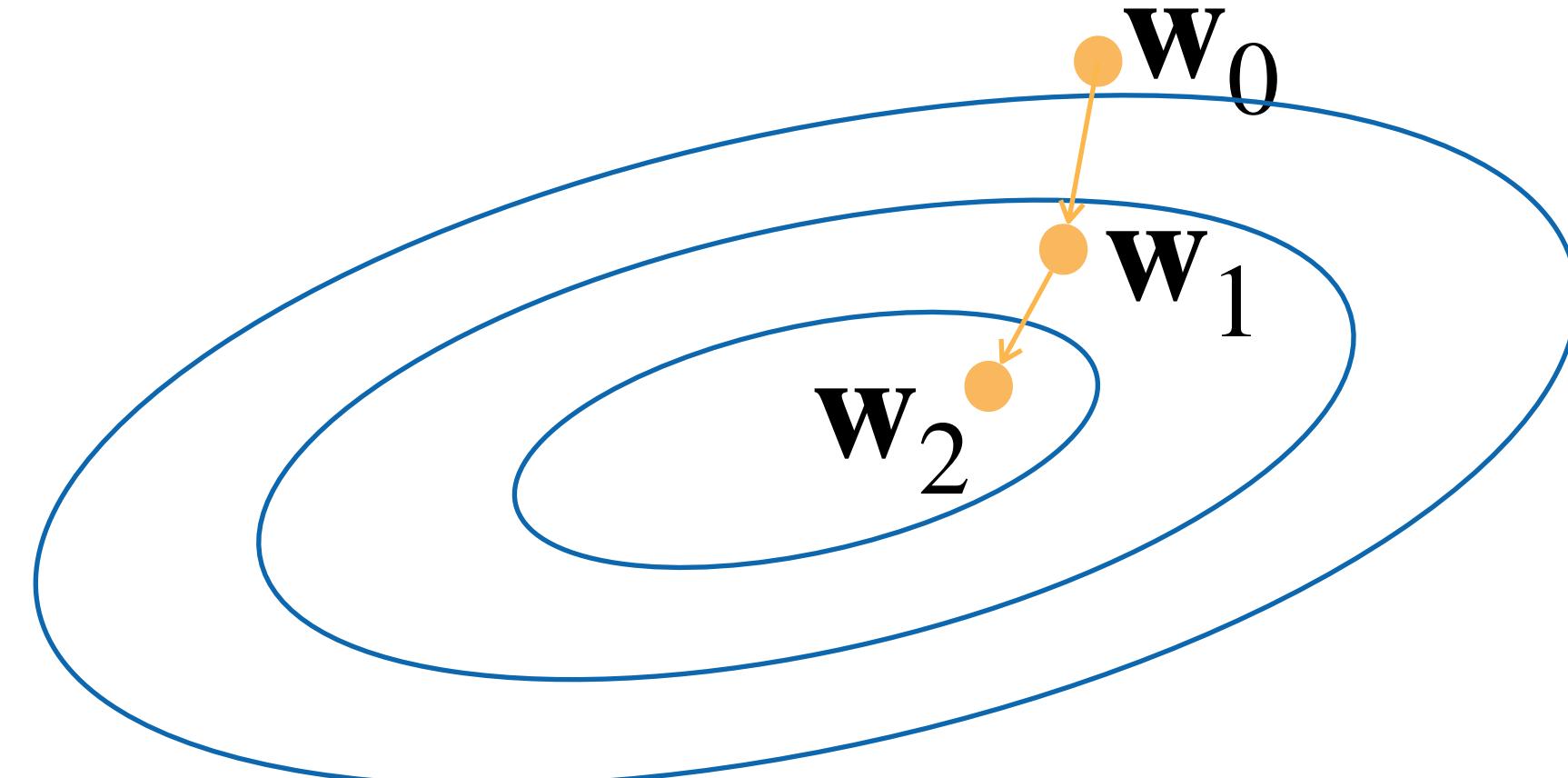
Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For $t = 1, 2, \dots$

- Update parameters:

$$\begin{aligned} \mathbf{w}_t &= \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}} \\ &= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}} \end{aligned}$$

- Repeat until converges

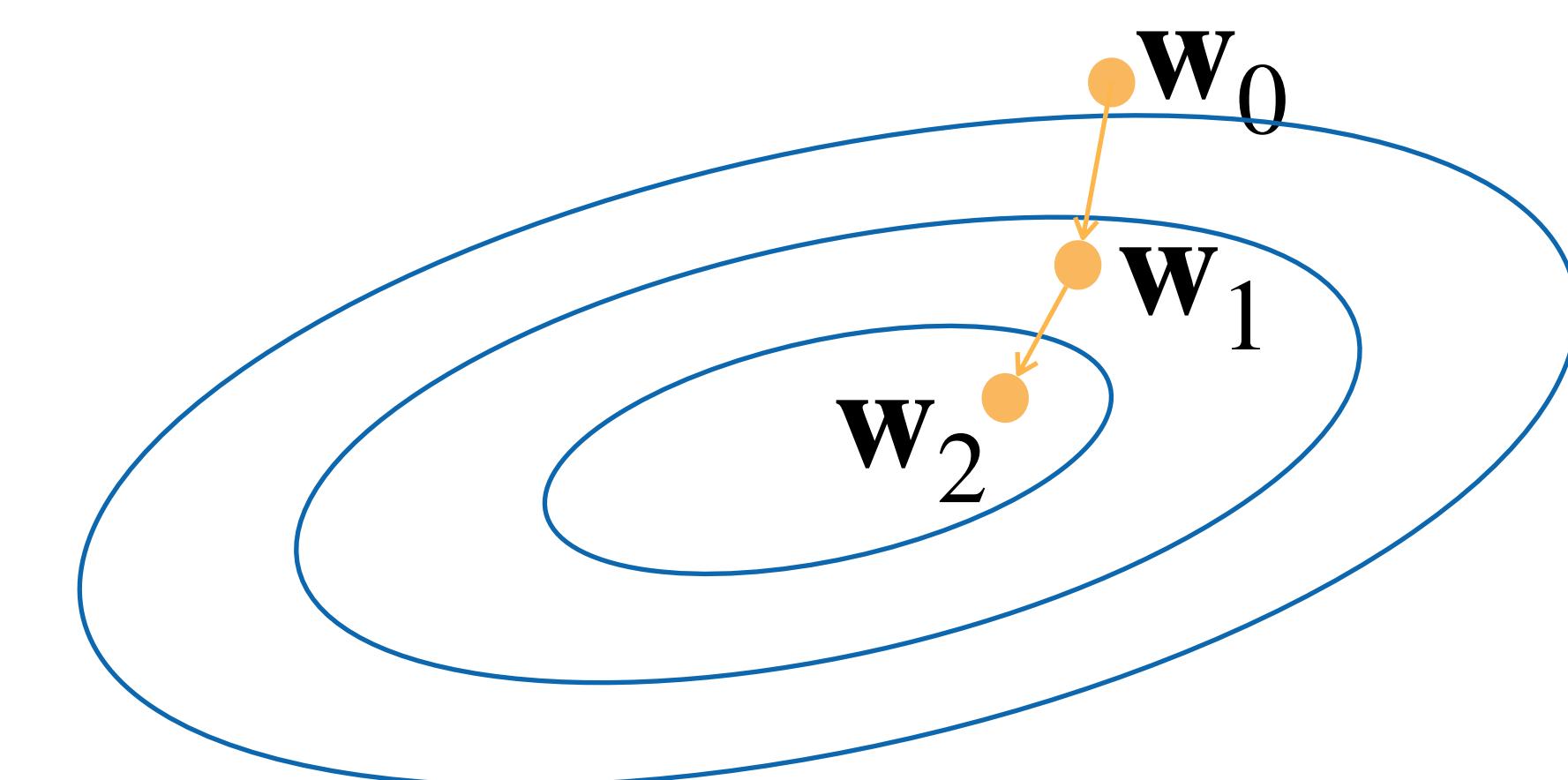


Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For $t = 1, 2, \dots$

- Update parameters:

$$\begin{aligned} w_t &= w_{t-1} - \alpha \frac{\partial L}{\partial w_{t-1}} \\ &= w_{t-1} - \alpha \frac{1}{|D|} \sum_{(x,y) \in D} \frac{\partial \ell(x, y)}{\partial w_{t-1}} \end{aligned}$$



D can
be very large.
Expensive

- Repeat until converges

Gradient Descent

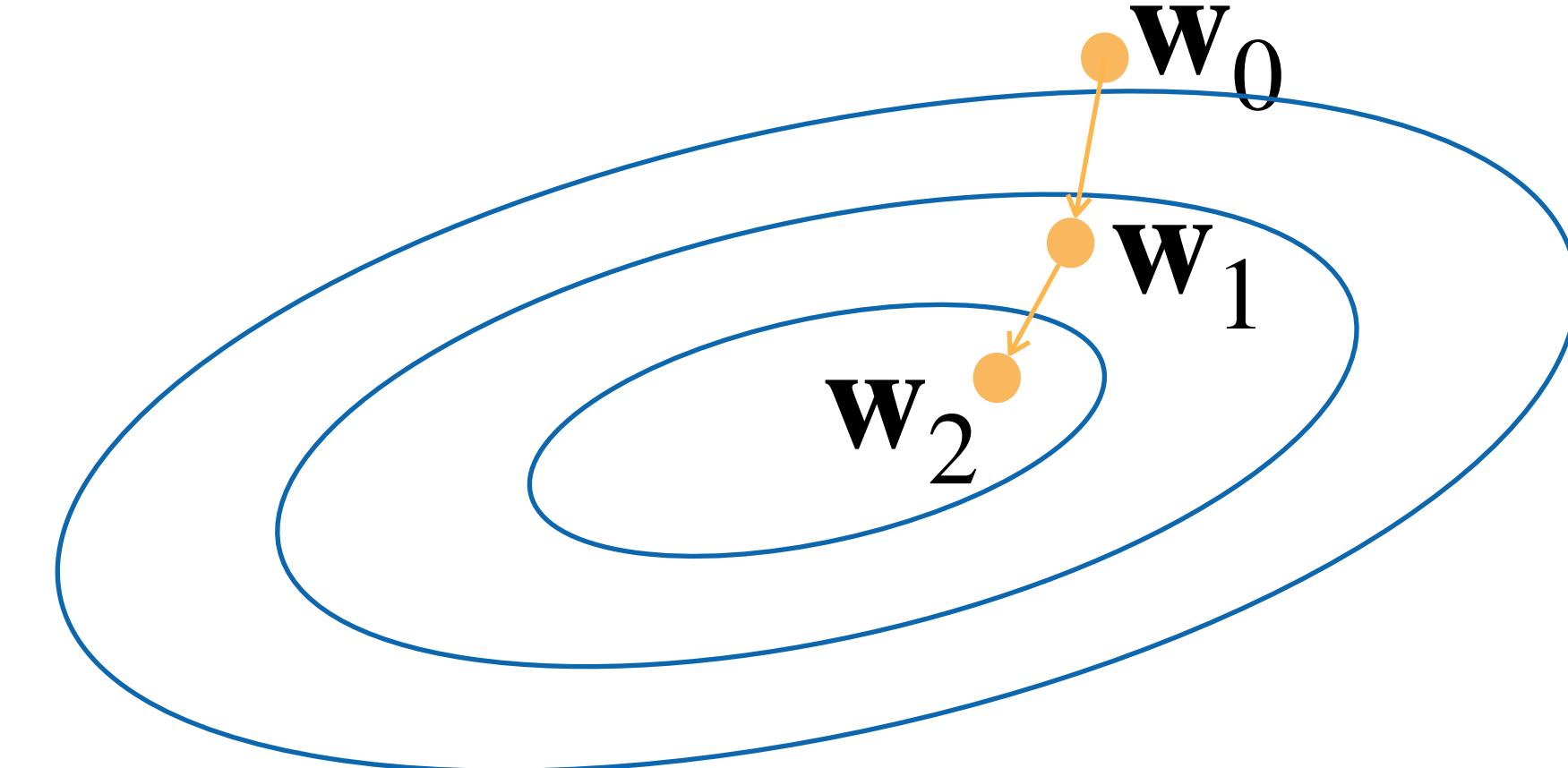
- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For $t = 1, 2, \dots$

- Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

$$= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges



D can
be very large.
Expensive

The gradient w.r.t. all
parameters is obtained by
concatenating the partial
derivatives w.r.t. each
parameter

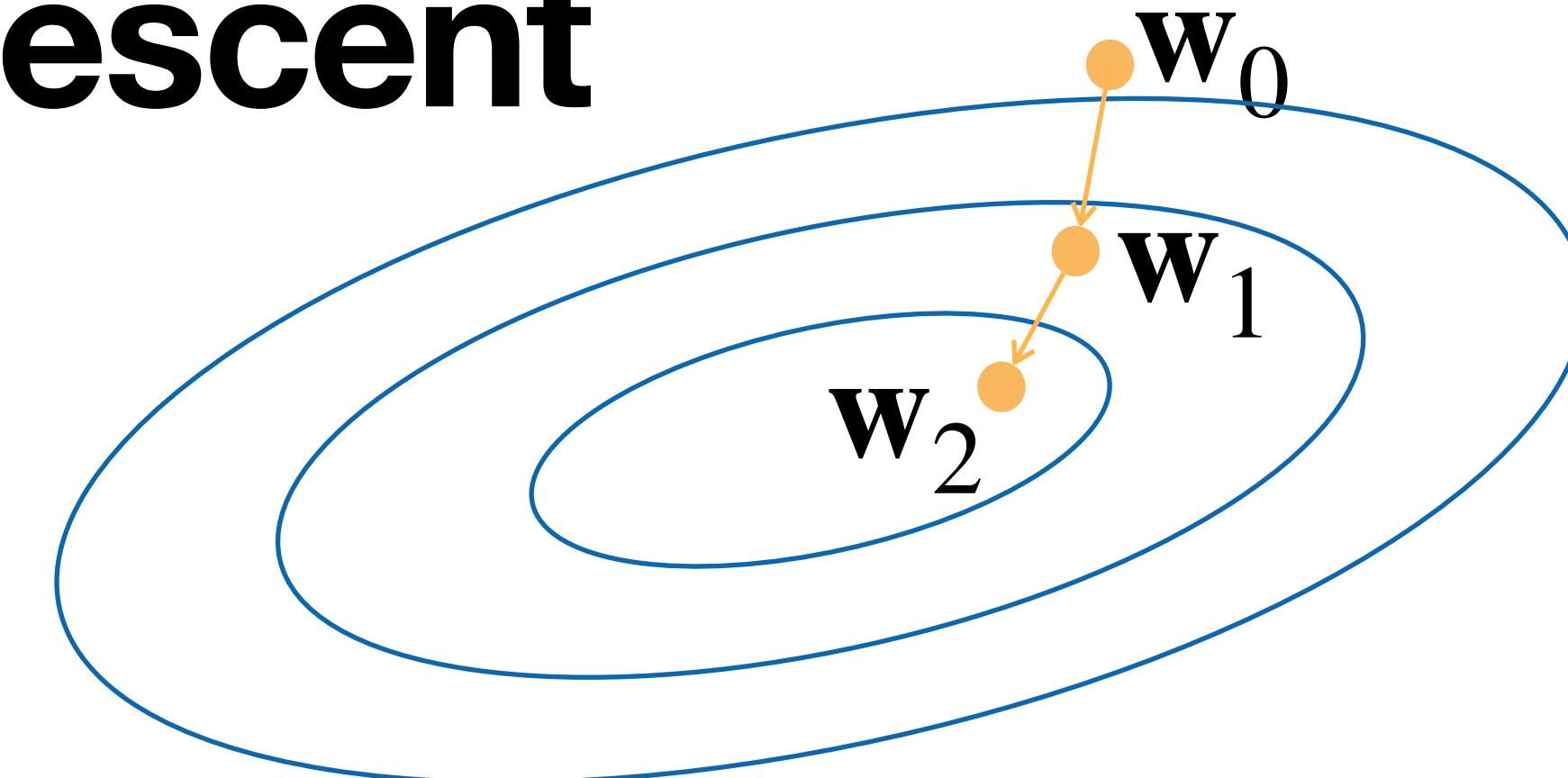
Minibatch Stochastic Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For $t = 1, 2, \dots$

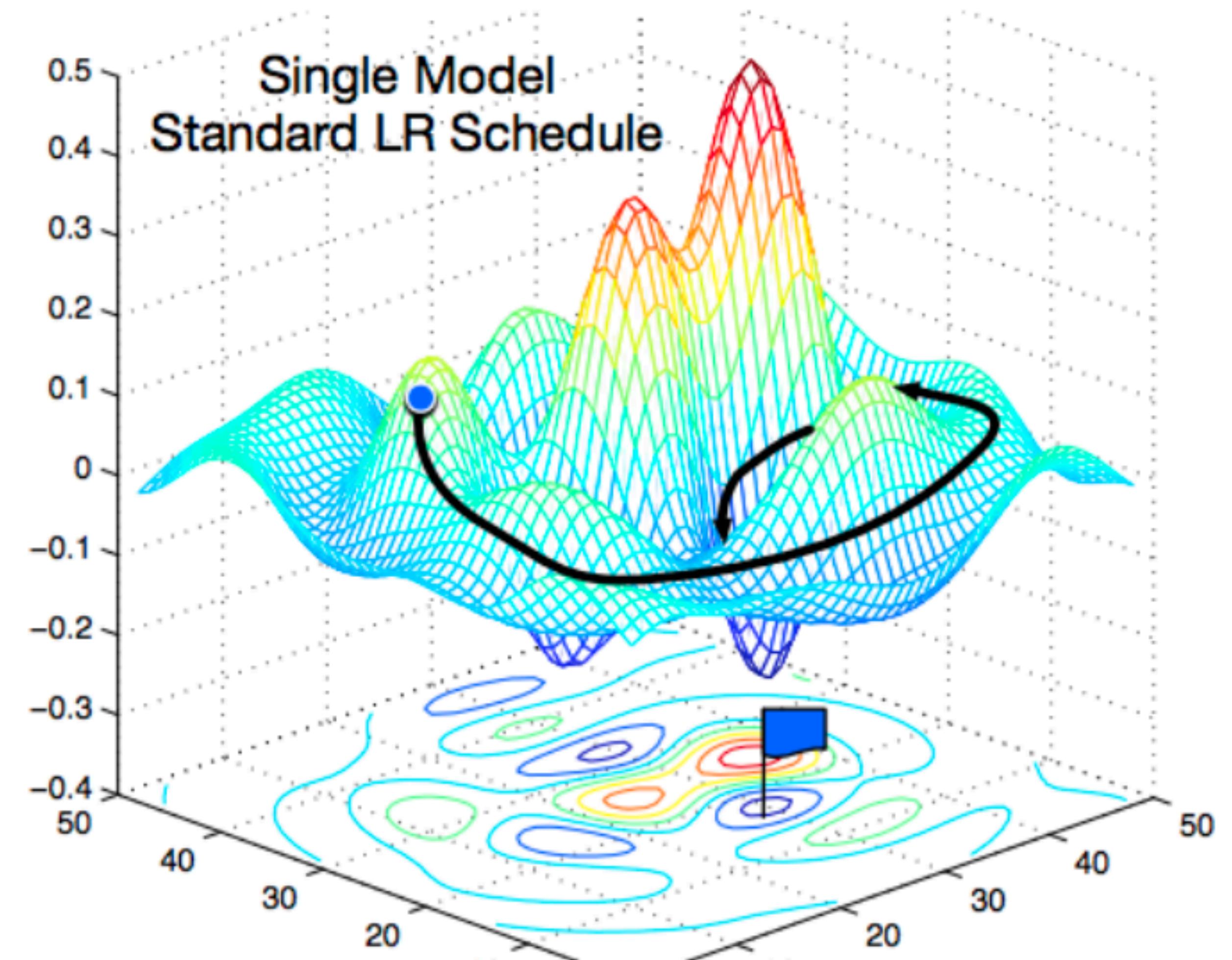
- Randomly sample a subset (mini-batch) $B \subset D$
Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

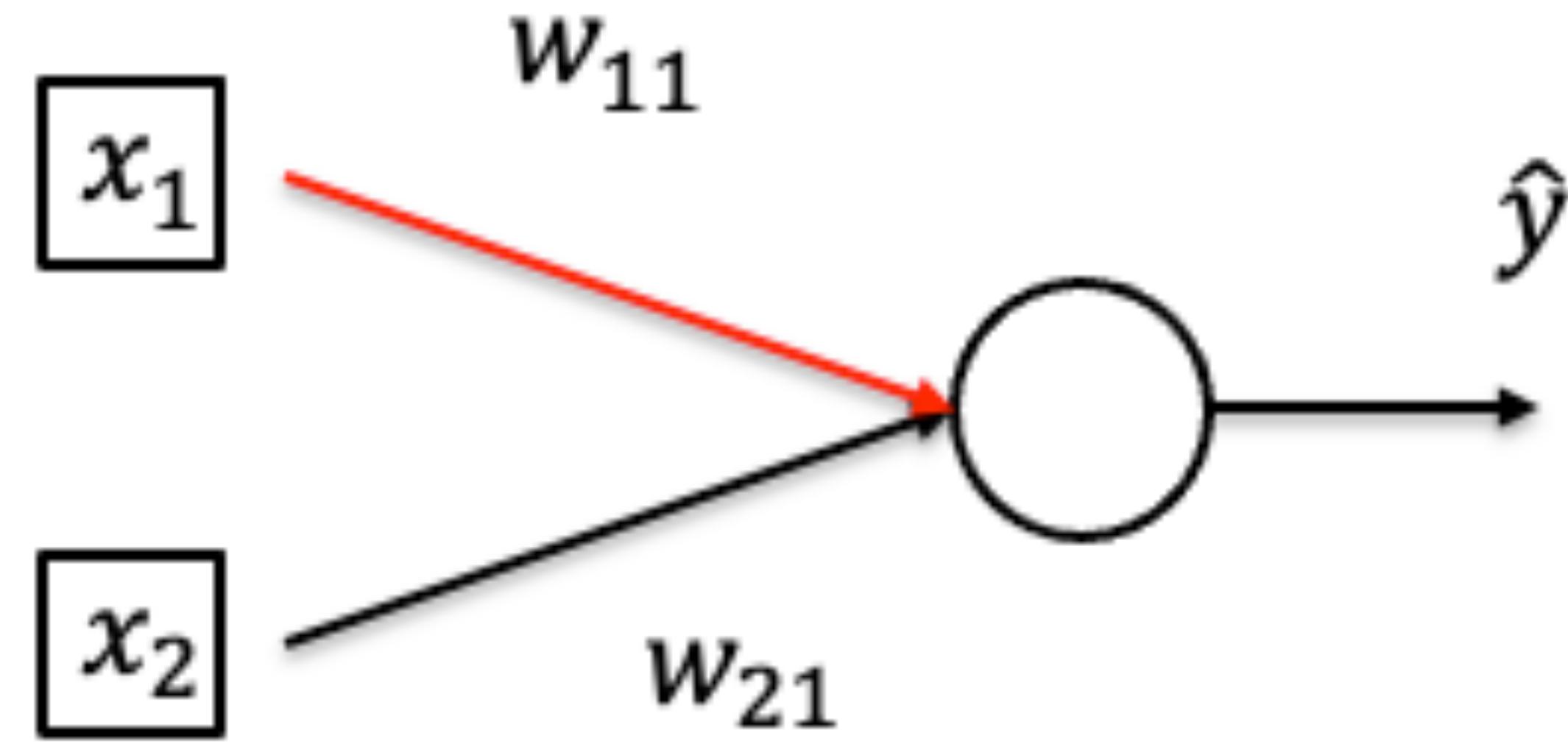


Non-convex Optimization



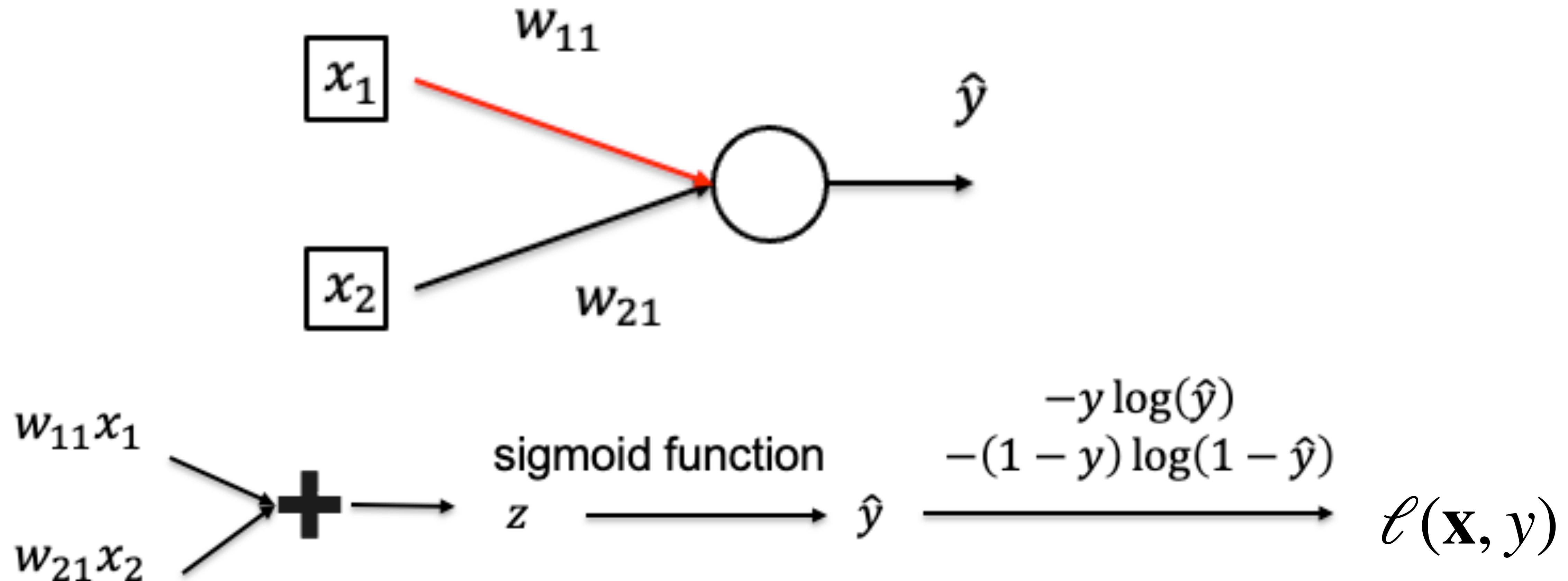
[Gao and Li et al., 2018]

Calculate Gradient (on one data point)

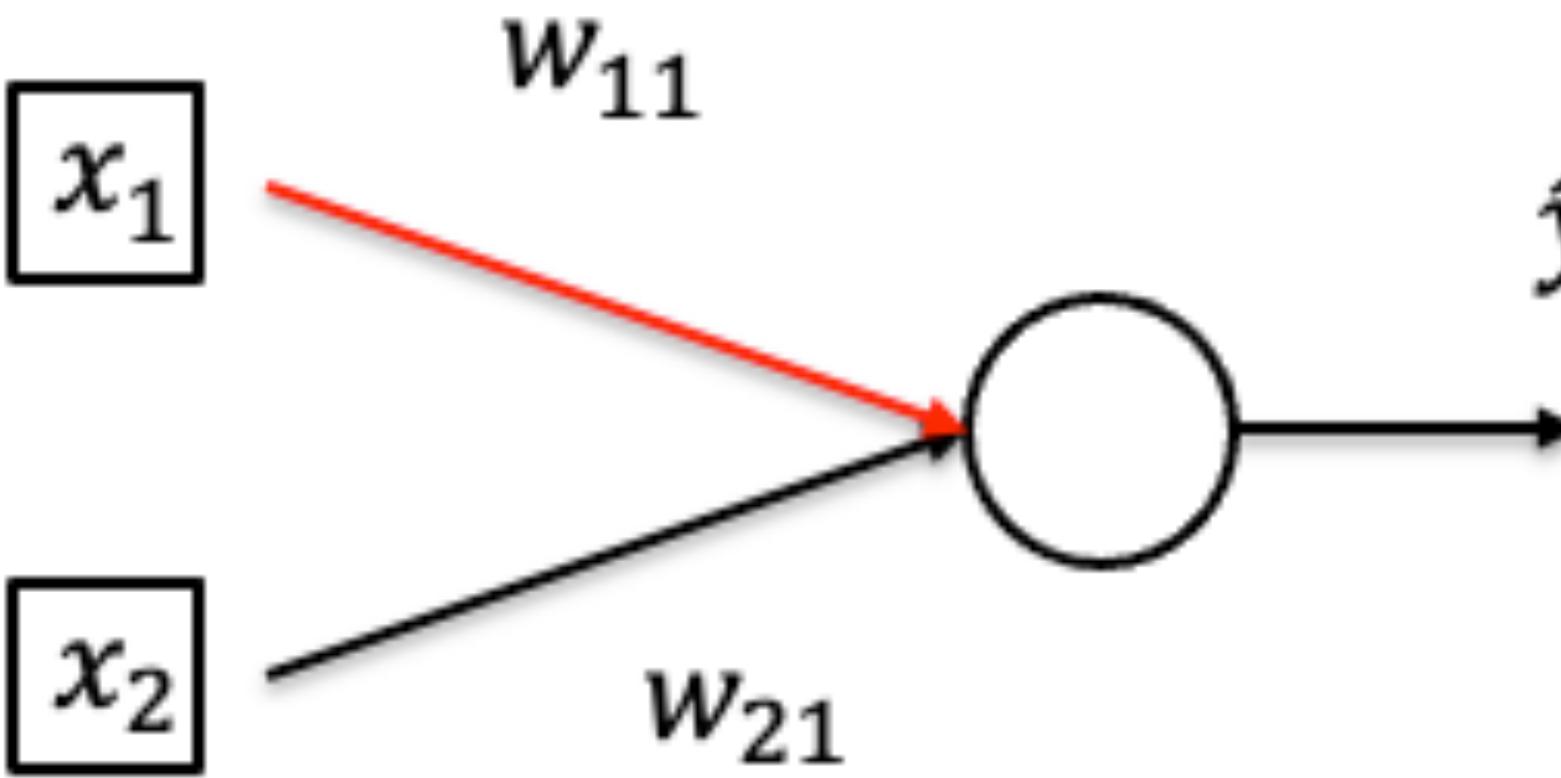


- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

Calculate Gradient (on one data point)



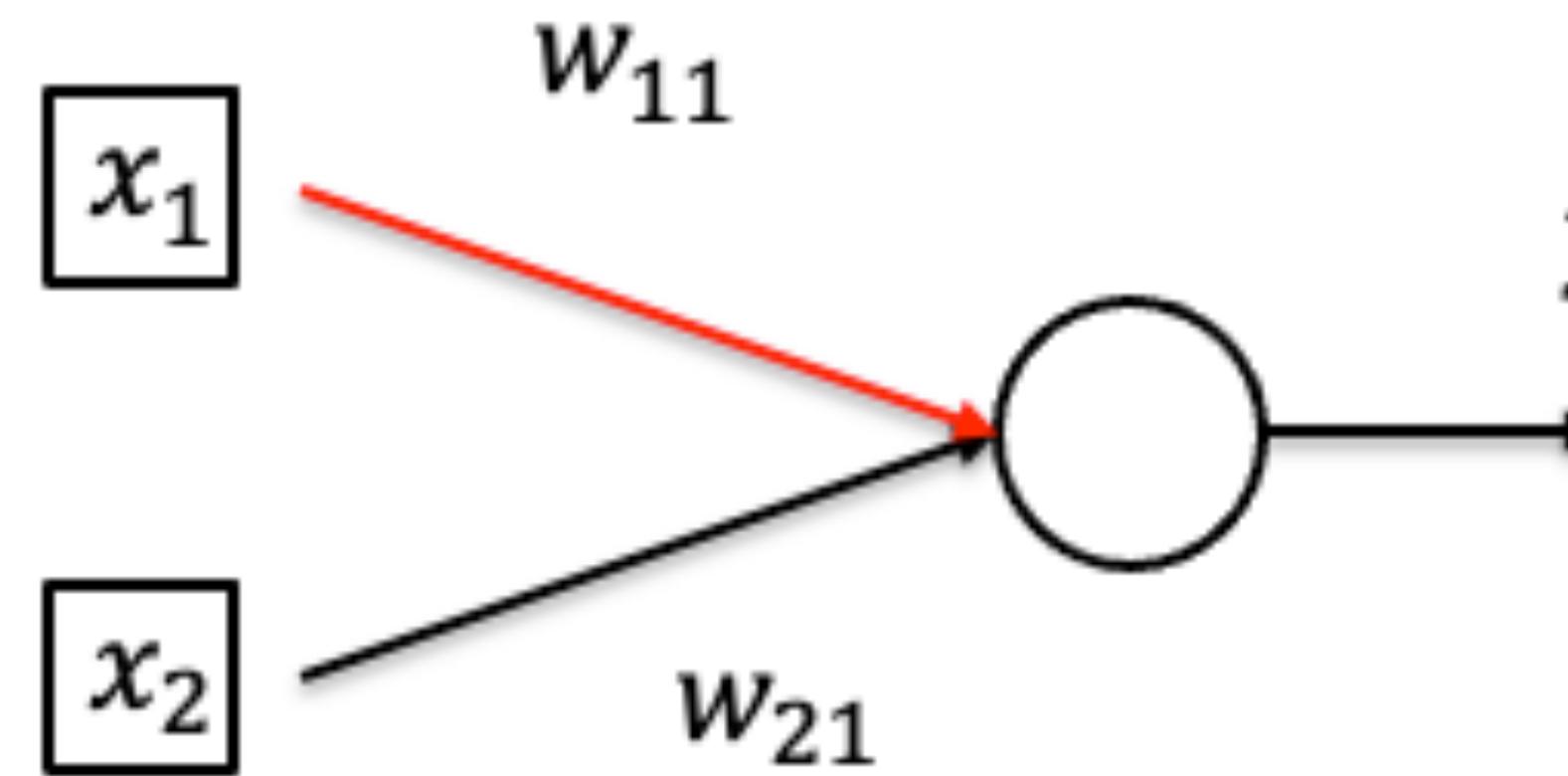
Calculate Gradient (on one data point)



$w_{11}x_1$ $w_{21}x_2$ $\rightarrow \text{+} \rightarrow z \rightarrow \text{sigmoid function} \rightarrow \hat{y} \rightarrow \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} \rightarrow \ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z)$$
$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$$

Calculate Gradient (on one data point)



A computational graph illustrating the forward pass and backpropagation for a logistic regression model. The forward pass consists of three main steps:

- Inputs $w_{11}x_1$ and $w_{21}x_2$ are summed to produce z .
- The sigmoid function is applied to z to produce the prediction \hat{y} .
- The loss function $\ell(\mathbf{x}, y)$ is calculated based on \hat{y} and the target y .

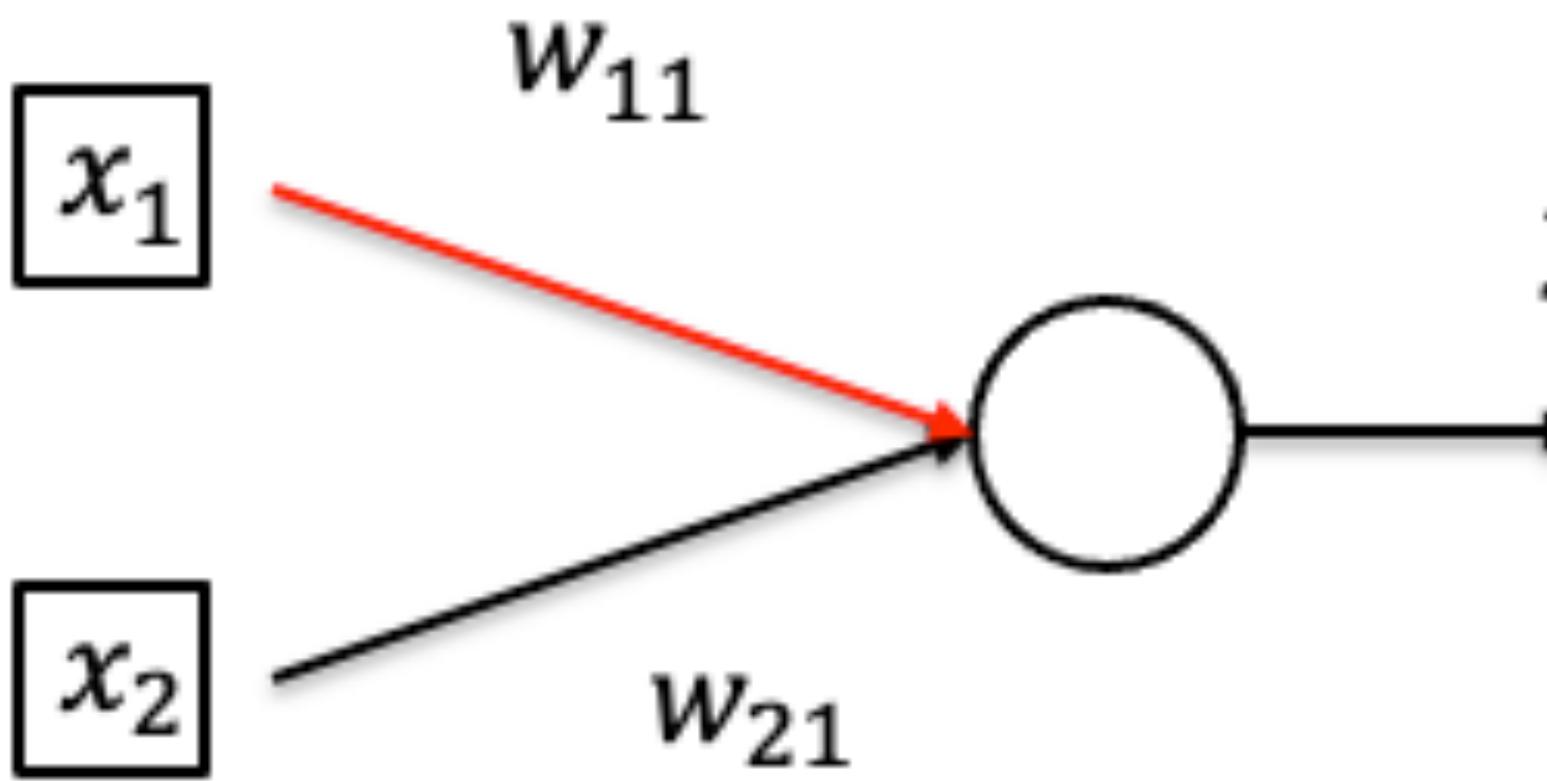
The backpropagation step shows the derivative of the loss with respect to the prediction \hat{y} :

$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)

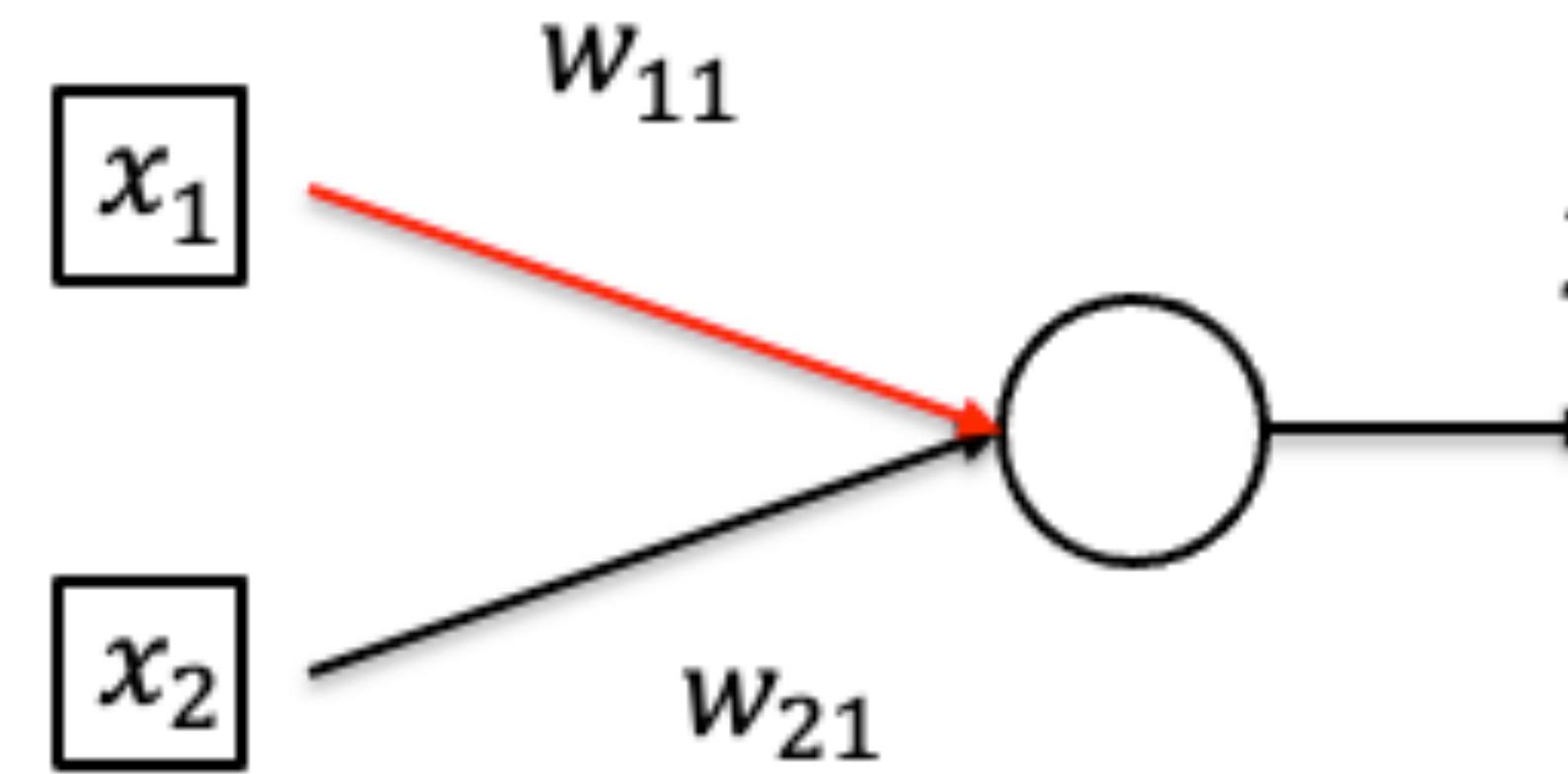


$$\begin{array}{ccccc} w_{11}x_1 & \xrightarrow{\text{+}} & z & \xrightarrow{\text{sigmoid function}} & \hat{y} \\ w_{21}x_2 & & & & \xrightarrow{-y \log(\hat{y})} \\ & & & & -(1 - \hat{y}) \log(1 - \hat{y}) \\ & & & & \ell(\mathbf{x}, y) \\ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) & & & \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} & \end{array}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1$$

Calculate Gradient (on one data point)



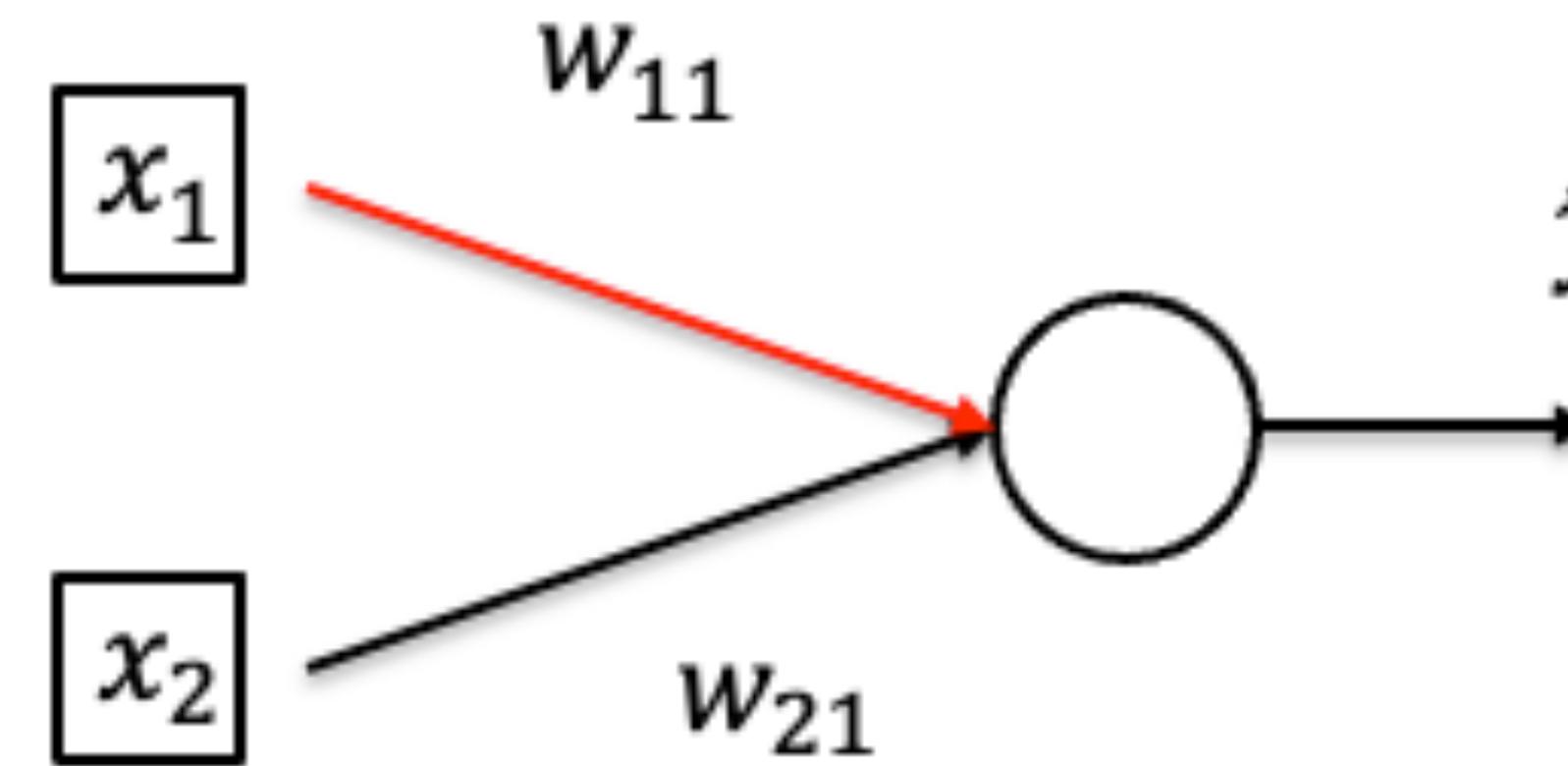
A computational graph illustrating the forward pass and the backward pass for calculating the gradient. The forward pass consists of three main steps: 1) Inputs $w_{11}x_1$ and $w_{21}x_2$ are summed at a plus node to produce z . 2) The sigmoid function maps z to \hat{y} . 3) The loss function maps \hat{y} to the loss value $\ell(\mathbf{x}, y)$. The backward pass involves calculating gradients: $\frac{\partial \ell}{\partial \hat{y}} = -\frac{y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})}{-(1 - \hat{y}) \log(1 - \hat{y})}$ and $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \text{+} \rightarrow z \xrightarrow{\text{sigmoid function}} \hat{y} \xrightarrow{-\frac{y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})}{-(1 - \hat{y}) \log(1 - \hat{y})}} \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \hat{y}(1 - \hat{y})x_1$$

Calculate Gradient (on one data point)



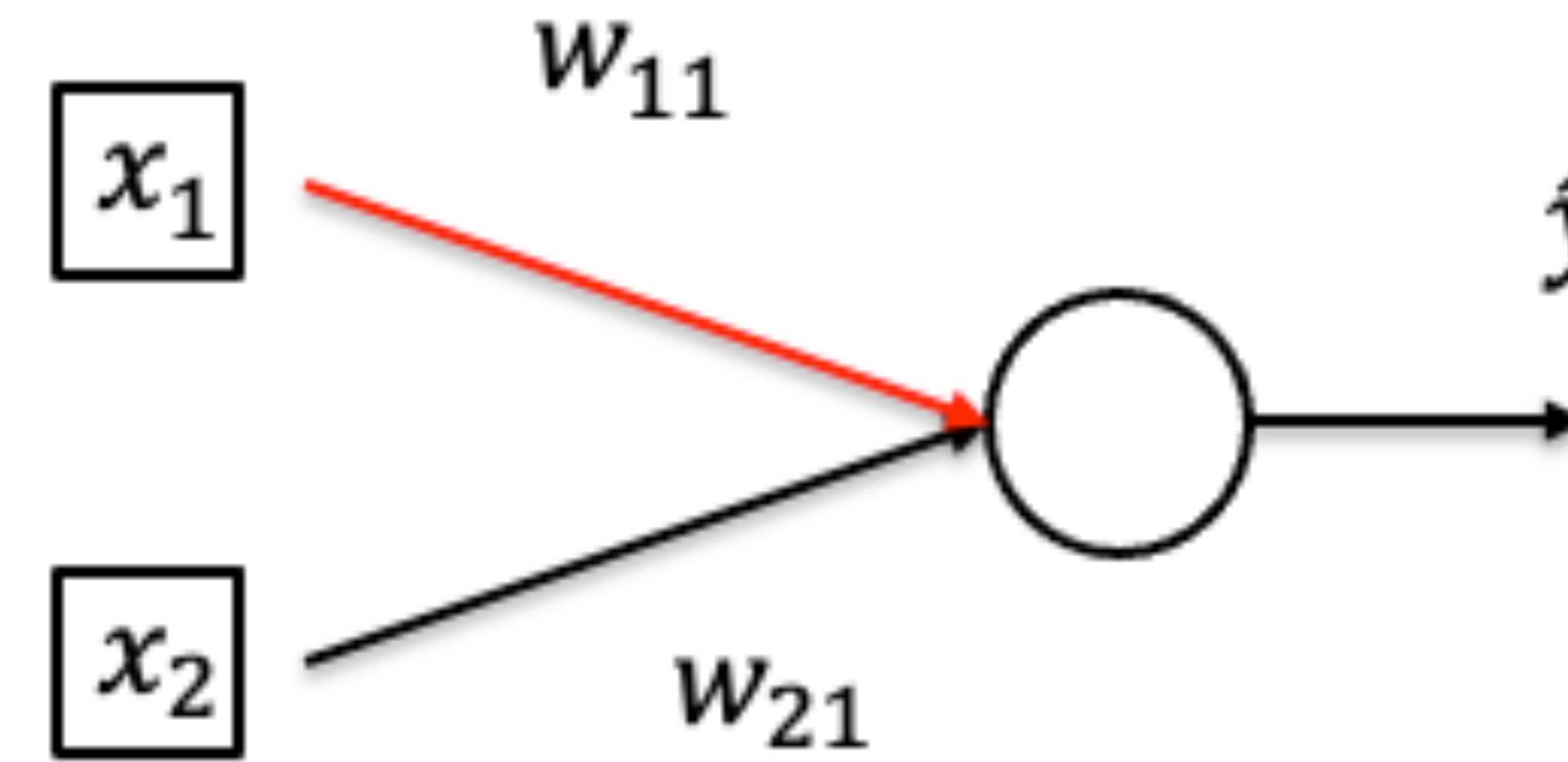
The diagram illustrates the forward pass of a neural network. It starts with inputs $w_{11}x_1$ and $w_{21}x_2$ which are summed at a plus sign. The result is passed through a "sigmoid function" to produce the output \hat{y} . The output \hat{y} is then used in the loss function $\ell(\mathbf{x}, y)$. Below the diagram, the derivative of the sigmoid function is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

$$\begin{array}{c} w_{11}x_1 \\ w_{21}x_2 \end{array} \rightarrow \text{+} \rightarrow \text{sigmoid function} \rightarrow z \rightarrow \hat{y} \rightarrow \ell(\mathbf{x}, y)$$
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) \hat{y}(1-\hat{y})x_1$$

Calculate Gradient (on one data point)



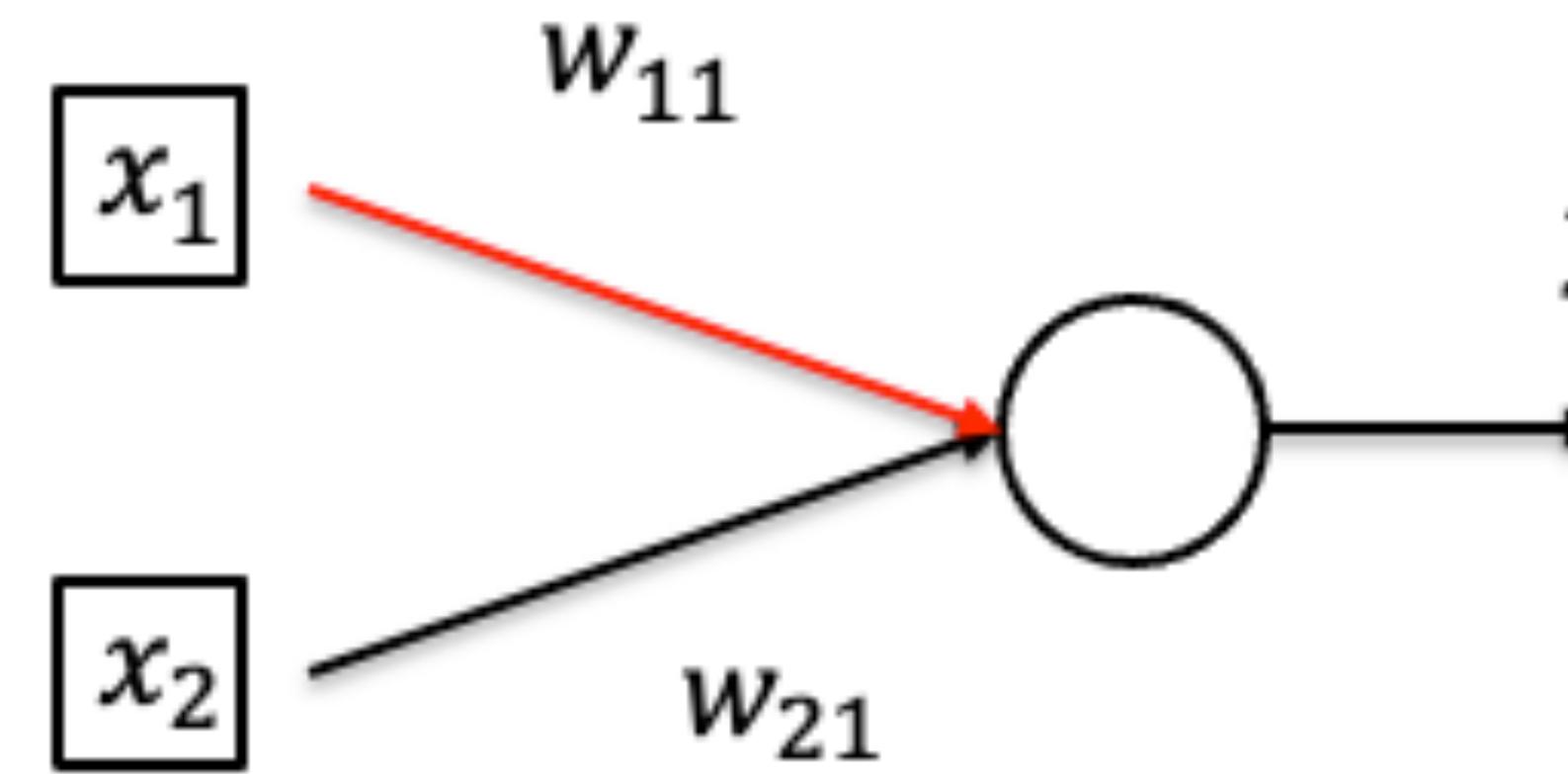
A computational graph illustrating the forward pass and loss calculation:

- Inputs $w_{11}x_1$ and $w_{21}x_2$ are summed at a plus node.
- The result of the summation is passed through a "sigmoid function".
- The output of the sigmoid function is \hat{y} .
- The loss function is calculated as $-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$.
- The derivative of the loss with respect to \hat{y} is shown as $\frac{\partial l}{\partial \hat{y}} = -(y/\hat{y}) + ((1 - y)/(1 - \hat{y}))$.
- The derivative of the sigmoid function with respect to z is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$$

Calculate Gradient (on one data point)

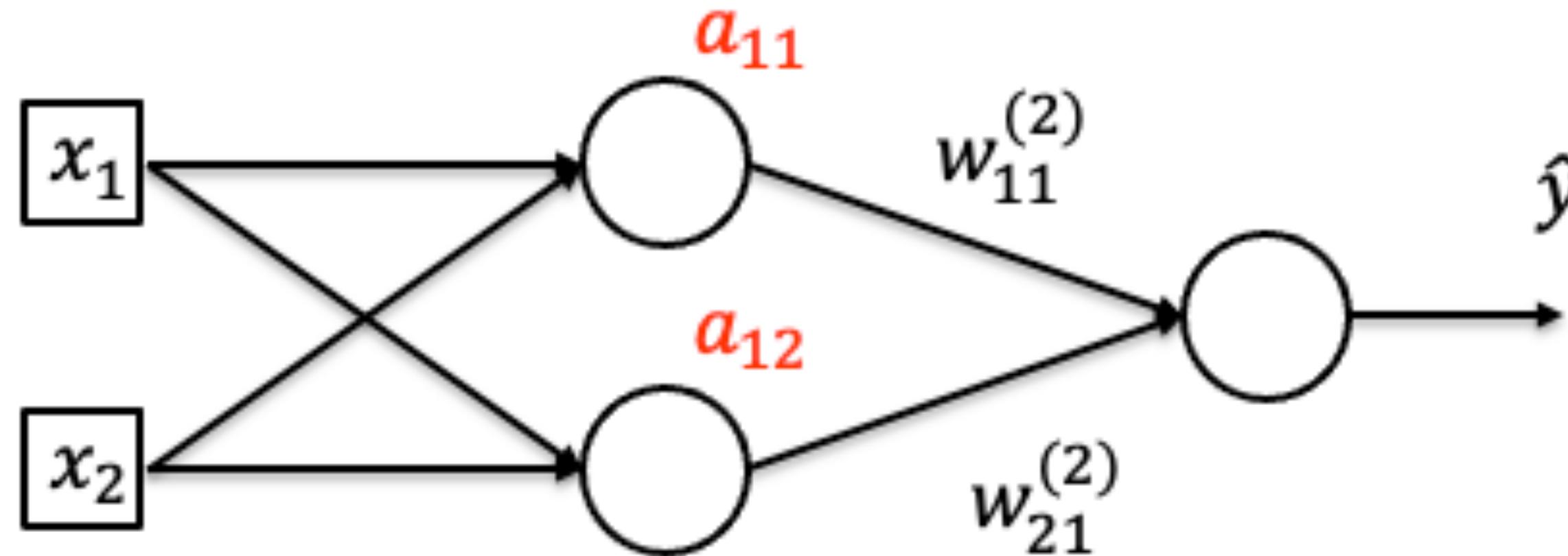


A computational graph illustrating the forward pass and the calculation of gradients. On the left, inputs $w_{11}x_1$ and $w_{21}x_2$ are summed to produce z . This value z is passed through a "sigmoid function" to produce the output \hat{y} . The output \hat{y} is then used in the loss function $\ell(\mathbf{x}, y)$, which is defined as $-\frac{-y \log(\hat{y})}{-(1 - y) \log(1 - \hat{y})}$. Below this, the derivative of the sigmoid function is given as $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$.

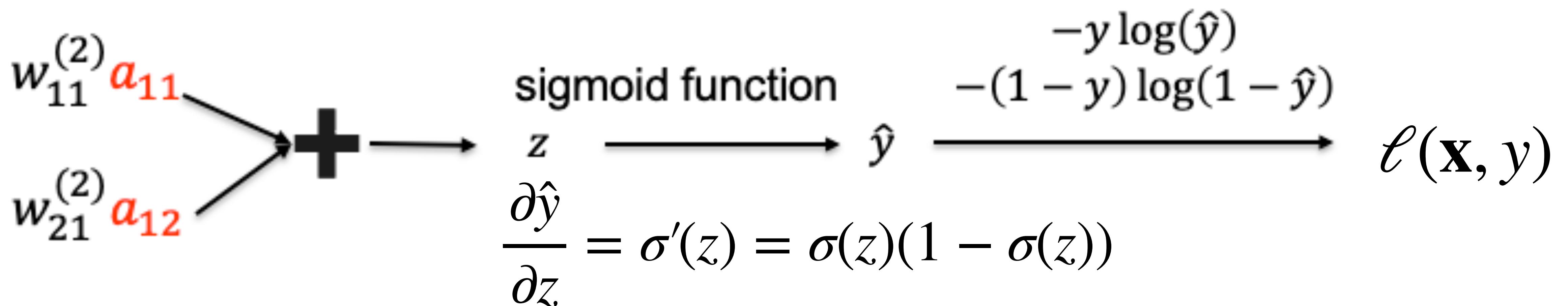
- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y)w_{11}$$

Calculate Gradient (on one data point)

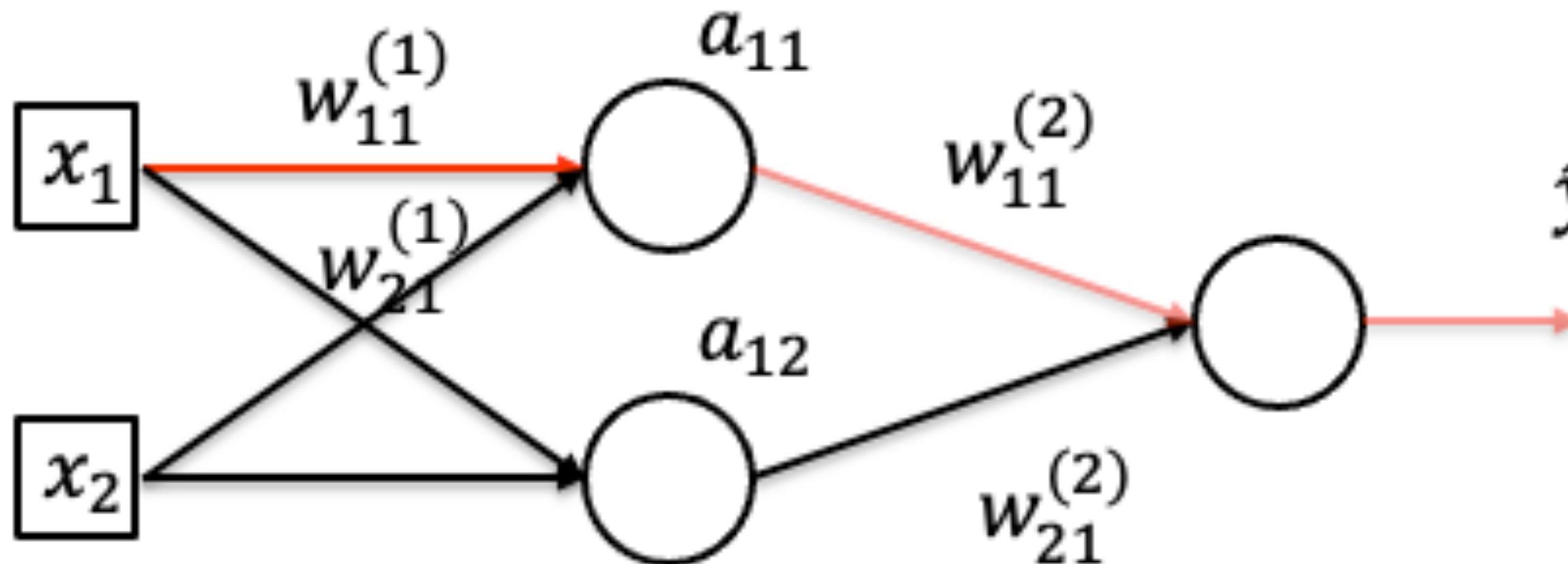


Make it deeper



- By chain rule: $\frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$, $\frac{\partial l}{\partial a_{12}} = (\hat{y} - y)w_{21}^{(2)}$

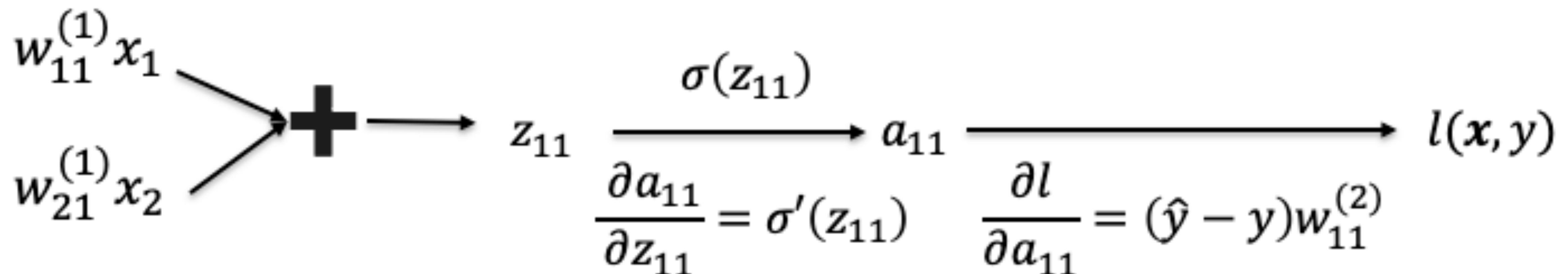
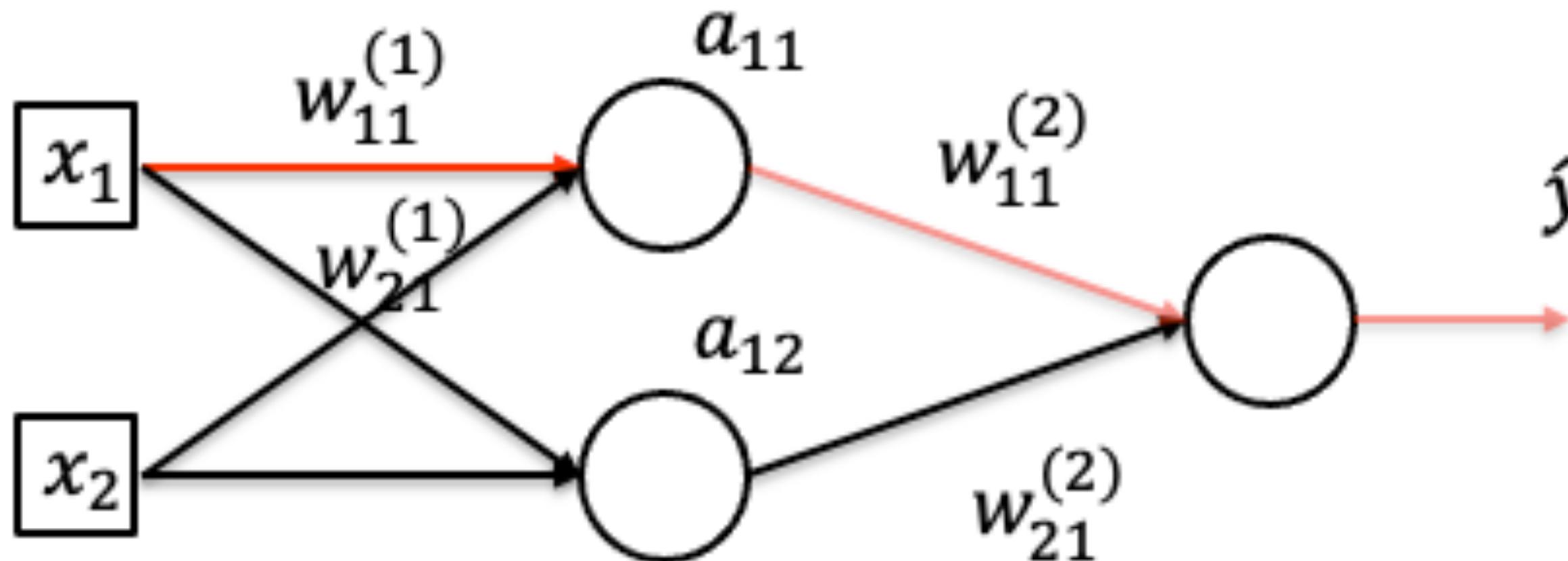
Calculate Gradient (on one data point)



$$\begin{aligned} w_{11}^{(1)} x_1 & \quad \quad \quad + \quad \quad \quad \sigma(z_{11}) & & l(x, y) \\ w_{21}^{(1)} x_2 & \quad \quad \quad z_{11} \quad \longrightarrow \quad a_{11} \quad \longrightarrow \quad \frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)} \end{aligned}$$

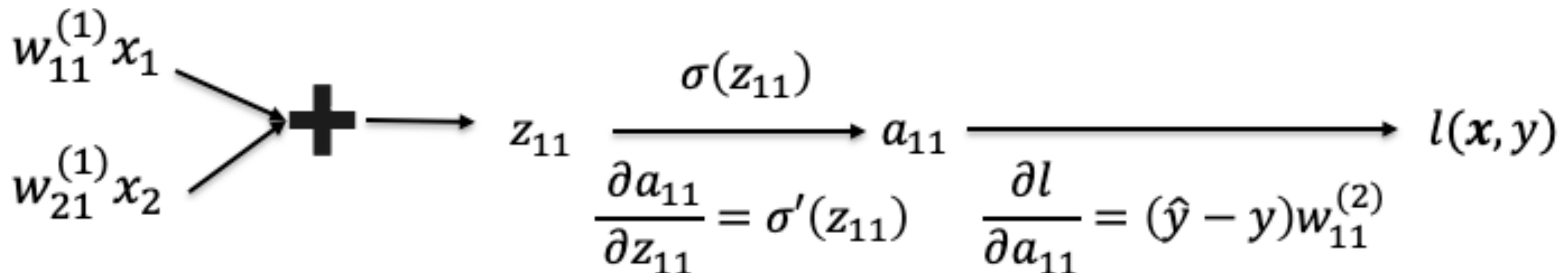
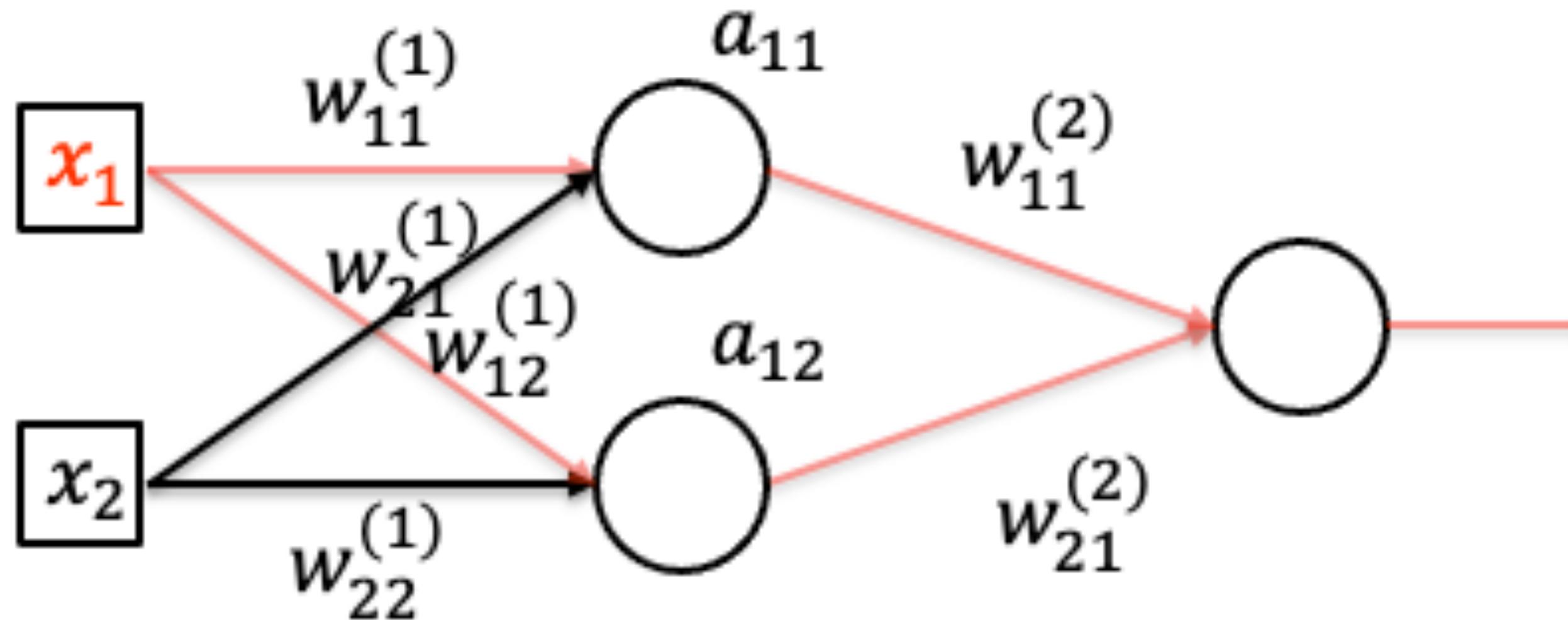
- By chain rule: $\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y) w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$

Calculate Gradient (on one data point)



- By chain rule: $\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)}a_{11}(1 - a_{11})x_1$

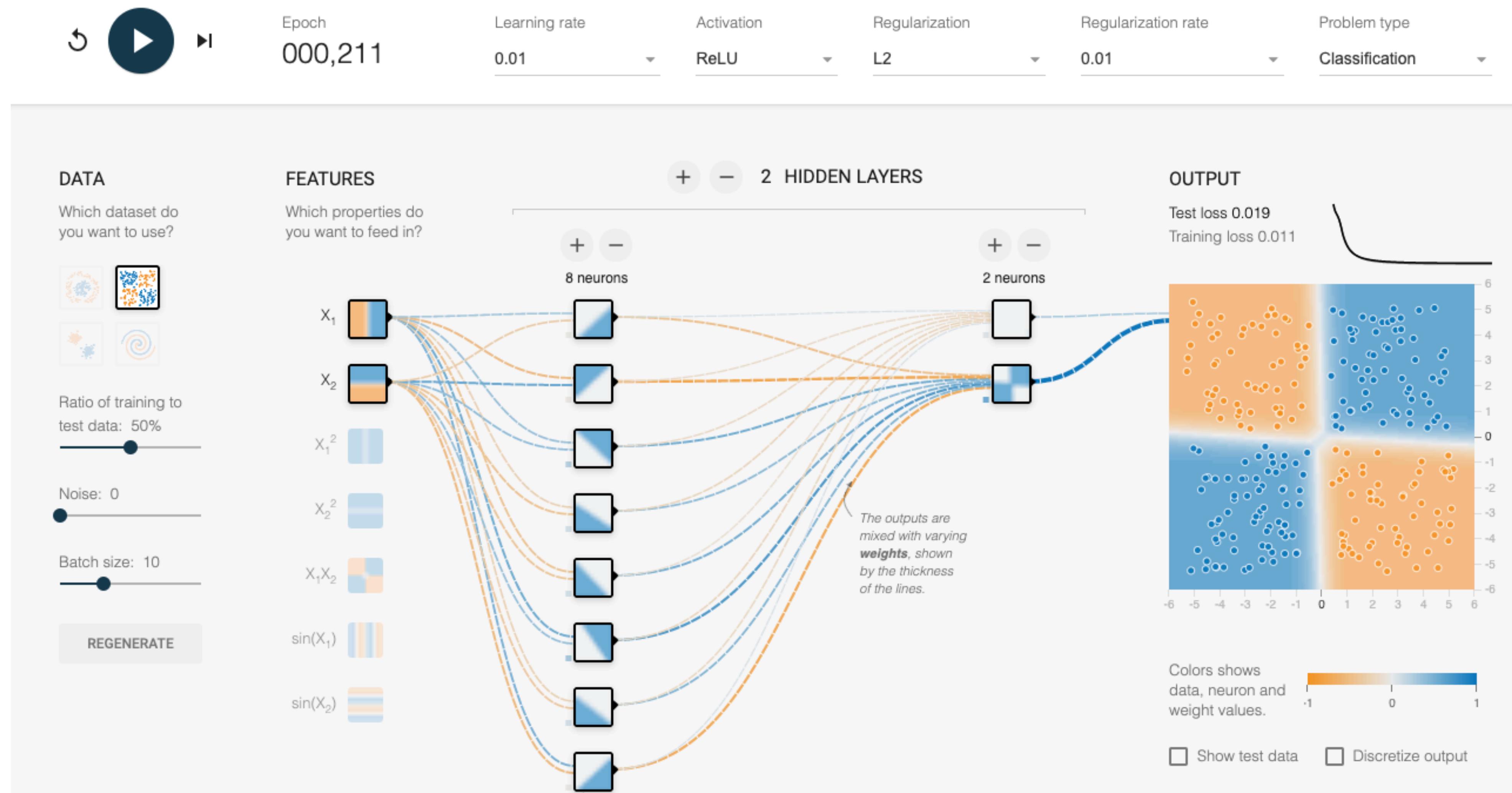
Calculate Gradient (on one data point)



- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}$$

Demo: Learning XOR using neural net



• <https://playground.tensorflow.org/>

What we've learned today...

- Single-layer Perceptron Review
- Multi-layer Perceptron
 - Single output
 - Multiple output
- How to train neural networks
 - Gradient descent